VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

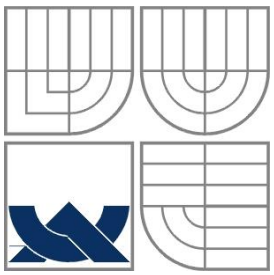
EFEKTIVNÍ ALGORITMY PRO PRÁCI S BÜCHIHO AUTOMATY

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

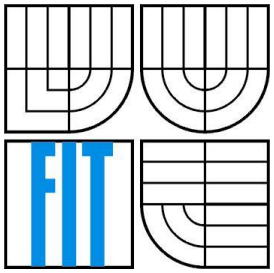
AUTOR PRÁCE
AUTHOR

TOMÁŠ LAŠČÁK

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

EFEKTIVNÍ ALGORITMY PRO PRÁCI S BÜCHIHO AUTOMATY

EFFICIENT ALGORITHMS FOR BÜCHI AUTOMATA

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR
TOMÁŠ LAŠČÁK

VEDOUCÍ PRÁCE
SUPERVISOR
ING. ONDŘEJ LENGÁL

BRNO 2014

Abstrakt

Cílem této práce je rozšířit existující knihovnu VATA o modul pro práci s Büchiho automaty, které se řadí mezi konečně stavové automaty nad nekonečnými slovy. Tyto automaty jsou využívány v různých oblastech informatiky, mimo jiné také ve formální verifikaci, při LTL model checkingu. LTL model checking se provádí typicky za pomocí operace testování jazykové inkluze mezi dvěma Büchiho automaty. Protože jazyková inkluze může být výpočetně velmi náročná, vzniklo několik optimalizovaných algoritmů pro tento problém, jako je například přístup založený na Ramseyho větě. Předkládaná práce je zaměřena na tento přístup, jehož implementace je přidána do nově vytvořeného rozšíření knihovny VATA. Mimo to jsou do tohoto nového rozšíření přidány také další operace nad Büchiho automaty, jako jsou sjednocení, průnik nebo redukce počtu stavů.

Abstract

The main goal of this work is to extend the existing library VATA with a module for working with Büchi automata, which are finite state automata over infinite words. These automata are used in many areas of computer science, among others in formal verifications, namely in LTL model checking. LTL model checking is typically carried out using the operation of testing language inclusion between two Büchi automata. Because testing language inclusion can be computationally very demanding, several optimized algorithms have been created for testing language inclusion such as the Ramsey-based approach. This work is focused on the mentioned approach, the implementation of which is added to the newly created extension of the VATA library. Apart from that, the new extension adds additional useful operations over Büchi automata such as union, intersection or a reduction in the number of states.

Klíčová slova

Büchiho automaty, formální verifikace, jazyková inkluze, knihovna VATA

Keywords

Büchi automata, formal verification, language inclusion, VATA library

Citace

Tomáš Laščák: Efektivní algoritmy pro práci s Büchiho automaty, bakalářská práce, Brno, FIT VUT v Brně, 2014

Efektivní algoritmy pro práci s Büchiho automaty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Ondřeje Lengála.

……
Tomáš Laščák
20. května 2014

Poděkování

Rád bych tímto poděkoval vedoucímu této práce, Ing. Ondřeji Lengálovi, za odborné rady a vedení při tvorbě práce.

Obsah

1 Úvod

V počítačové vědě a teorii formálních jazyků zaujímají Büchiho automaty důležité místo. Jedná se o modifikaci konečně stavových automatů, které pracují nad nekonečně dlouhými vstupními slovy. Patří do třídy automatů nad nekonečnými slovy, tzv. ω-automatů a poprvé je definoval v roce 1962 švýcarský matematik Julius Richard Büchi v [1], po kterém jsou také tyto automaty pojmenovány. Büchiho automaty bývají především využívány v oblasti formální verifikace.

Knihovna VATA [3] je vysoce optimalizovaná knihovna (napsána v jazyce C++) pro práci se stromovými konečnými automaty, více v kapitole 3.2. Cílem této práce je rozšířit tuto knihovnu o možnost práce s Büchiho automaty, včetně podpory některých operací. Konkrétně se jedná o podporu efektivních algoritmů pro sjednocení, průnik, redukci počtu stavů a testování jazykové inkluze.

V kapitole 2 je stručný popis teorie automatů nad nekonečnými slovy a to převážně Büchiho automatů. Jsou zde uvedeny některé známé typy automatů nad nekonečnými slovy. Dále se zde také setkáme s teoretickým popisem ω-regulárních jazyků a k nim příslušným uzávěrovým vlastnostem, pro něž budou v této práci implementovány operace. Kapitola 3 popisuje již existující knihovny pro práci s těmito automaty a také knihovnu VATA, která bude během této práce využívána a rozšiřována o možnost práce s Büchiho automaty. Dále se v této kapitole nachází popis grafického nástroje GOAL a nástroje pro testování jazykové inkluze RABIT.

Cílem této práce není pouze rozšíření knihovny VATA o základní operace pro práci s Büchiho automaty, ale také o složitější operace. Jedná se o testování jazykové inkluze, k čemuž je v této práci využit přístup založený na Ramseyho větě. Teorie k tomuto algoritmu je zpracována v kapitole 4 a v kapitole 6 je popis implementace tohoto přístupu.

Kapitola 5 je zaměřena na popis implementace. Nejprve se zde nachází obeznámení s implementací datových struktur pro uložení jednotlivých částí Büchiho automatů. Dále jsou zde popsány základní operace, jako jsou sjednocení, průnik, reverze automatu, nebo také odstranění neukončujících a nedostupných stavů. Dozvíme se zde také, jak vypadá vstupní formát automatu, což je zároveň i jeho výstupní formát. Poté, co jsou popsány základní přístupy, se přechází k popisu implementace složitější části této práce. Jedná se o popis implementace převodu Büchiho automatu do LTS a jazykové inkluze.

Kapitola 7 pojednává o experimentech nad testováním jazykové inkluze a porovnání implementace přístupu ve zde prezentované práci a nástroje RABIT. Poslední částí je závěr, kde se nachází shrnutí dosažených výsledků této práce a diskutují se možná budoucí rozšíření a směry, kterými by se mohla práce dále ubírat.

2 Teorie automatů nad nekonečnými slovy

Cílem této práce je efektivní implementace algoritmů nad Büchiho automaty. Nejprve v této kapitole budou uvedeny základy teorie konečně stavových automatů nad nekonečnými slovy [2], tzv. ω-automatů, a jejich variant. Dále bude definována třída ω-regulárních jazyků a uzávěrové vlastnosti této třídy.

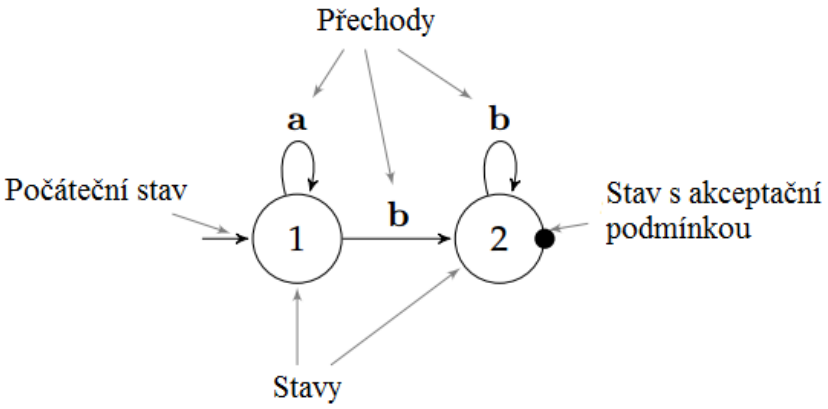
2.1 Jazyky nekonečných slov

Při práci s automaty nad nekonečnými slovy, musí být nejprve definován jejich jazyk. Nechť je Σ konečná množina symbolů nazývaná také abeceda. Σ^* je množina všech konečných slov nad Σ. Každé konečné slovo má délku, ta je definována máme-li slovo $x = a_1 \dots a_n$, pak jeho délka je $|x| = n$, pro $n \geq 1$ a $a_i \in \Sigma$ pro všechna $a_i \in \Sigma$ a $i = 1, \dots, n$. Opakem jsou nekonečná slova neboli tzv. *ω-slova*. Množina všech ω-slov nad abecedou Σ se značí jako Σ^ω.Tedy *ω-jazyk* $\mathcal{L}$ nad abecedou Σ je podmnožina Σ^ω tj. $\mathcal{L} \subseteq \Sigma^\omega$. Nejrozšířenější podtřídou z ω-jazyků je třída ω-regulárních jazyků, které mají tu užitečnou vlastnost, že jsou rozpoznatelné konečně stavovými automaty.

2.2 ω-automaty

ω-automat je variace konečného automatu, jehož vstupem není jako u konečného automatu konečný řetězec, ale jedná se o řetězec nekonečný. Tyto automaty jsou vhodné pro určení chování systémů, které běží nepřetržitě dlouho, jako je hardware, operační systémy či řídicí systémy.

Do třídy ω-automatů patří mimo jiné *Büchiho automaty, Rabinovy automaty, Streettovy automaty, paritní* a *Mullerovi automaty*, z nichž každý může být buďto deterministický nebo nedeterministický. Rozdíl mezi těmito automaty je v akceptační podmínce. Na Obr. 2.1 je příklad ω-automatu.



Obr. 2.1 - Příklad ω-automatu

2.2.1 Nedeterministický ω-automat

Hlavním účelem ω-automatu je definovat jazyk, což je podmnožina množiny všech vstupů. Zatímco u konečného automatu běh skončí v nějakém stavu, a pokud je tento stav koncovým, je vstupní slovo přijato. U ω-automatů je to složitější. Zde je vstup přijat, jestliže se odpovídající běh nachází v množině A_{acc}. Množina přijatých vstupních ω-slov se nazývá rozpoznaný *ω-jazyk* automatu a označuje se jako $\mathcal{L}(\mathcal{A})$.

Nedeterministický ω-automat je pětice $\mathcal{A} = (Q, \Sigma, \Delta, I, A_{acc})$ kde:

- Q je konečná množina stavů,
- Σ je vstupní abeceda,
- $\Delta \subseteq Q \times \Sigma \times Q$ je přechodová relace, $(q, a, p) \in \Delta$ se značí jako $q \xrightarrow{a} p$. Pro účely využití v některých algoritmech, které se nacházejí v následujících kapitolách, definujeme $\Delta(p, a) = \{q \in Q \mid p \xrightarrow{a} q\}$,
- $I \subseteq Q$ je množina počátečních stavů,
- A_{acc} je akceptační podmínka, $A_{acc} \subseteq Q^{\omega}$.

Nedeterministický ω-automat může přijmout mnoho různých běhů na daném vstupu nebo také nemusí přijmout žádný. Vstup je přijat, jestliže existuje alespoň jeden běh, který tento vstup přijme. Definice běhů a přijetí pro deterministické ω-automaty jsou pak speciálními případy nedeterministických případů.

Běh automatu $\mathcal{A}$ nad nekonečným slovem $\alpha = \alpha_1.\alpha_2.\alpha_3 \ldots$, kde $\alpha_i \in \Sigma$, $i \in \mathbb{N}$, je funkce $b : \mathbb{N}_0 \to Q$ pro niž platí:

- $b(0) \in I$,
- $b(i+1) \in b(i) \xrightarrow{\alpha+1} b(i+1), \forall i \in \mathbb{N}$,
- pro běh ρ značí $Inf(\rho)$ množinu stavů, které se v ρ vyskytují nekonečněkrát.

Cesta automatu $\mathcal{A}$ nad konečným slovem $\alpha = \alpha_1.\alpha_2.\alpha_n$ kde $\alpha_i \in \Sigma$ a $0 < i \leq n$ je konečná posloupnost $q_0 q_0 \ldots q_n$, kde $q_{j-1} \xrightarrow{a_j} q_j$ pro všechny $0 < j \leq n$. Cesta je akceptující tehdy a jen tehdy, pokud $q_i \in F$, pro některé, kde $0 < i \leq n$. Definujeme následující predikáty pro stavy $p, q \in Q$:

- $p \overset{w}{\Rightarrow}_F q$ tehdy a jen tehdy, pokud existuje akceptující cesta na w z p do q,
- $p \overset{w}{\Rightarrow} q$ tehdy a jen tehdy, pokud existuje cesta na w z p do q,
- $\neg \left(p \overset{w}{\Rightarrow} q\right)$ tehdy a jen tehdy, pokud neexistuje cesta na w z p do q.

Akceptující běh automatu $\mathcal{A}$ nad slovem α je takový běh, při kterém automat nekonečněkrát projde alespoň jedním z akceptujících stavů. Automat $\mathcal{A}$ akceptuje slovo α tehdy a jen tehdy, když pro něj existuje akceptující běh. Množinu slov akceptovaných automatem $\mathcal{A}$ značíme $\mathcal{L}(\mathcal{A})$. Jedná se o *jazyk automatu* $\mathcal{A}$. Důležitými pojmy jsou také

nedostupné a *neukončující stavy*. Nedostupný stav q automatu $\mathcal{A}$ je stav, pro který neexistuje běh $r = q_0 \dots q_n$, kde $q = q_n$ automatu $\mathcal{A}$ nad slovem $w \in \Sigma^*$ tak, že $q_0 \in I$. Neukončující stav q automatu $\mathcal{A}$ je stav, pro který neexistuje běh $r = q \dots q_n$ automatu $\mathcal{A}$ nad slovem $w \in \Sigma^*$.
Speciálním případem nedeterministického ω-automatu je deterministický ω-automat.
Deterministický ω-automat je pětice $\mathcal{A} = (Q, \Sigma, \delta, I, A_{acc})$ kde:

- Q je konečná množina stavů,
- Σ je vstupní abeceda,
- $\delta: Q \times \Sigma \to Q$ je přechodová funkce,
- $I \in Q$ je počáteční stav automatu,
- A_{acc} je akceptační podmínka, $A_{acc} \subseteq Q^{\omega}$.

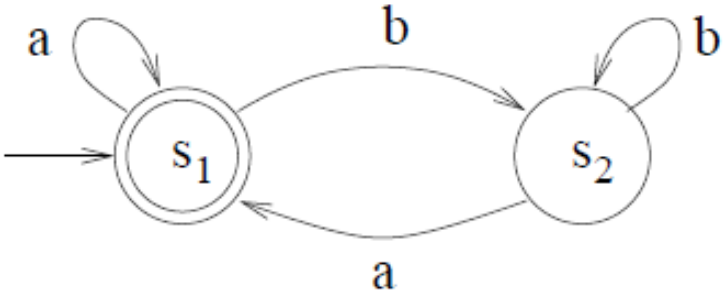
2.3 Büchiho automat

Büchiho automat (dále BA) je jeden z typů ω-automatu. Akceptační podmínka je množina stavů a BA přijímá vstupní posloupnost tehdy a jen tehdy, když existuje běh automatu, který navštíví alespoň jednou jeden stav z F nekonečně často. BA dokáže rozpoznat *ω-regulární jazyky*, což jsou regulární jazyky nad nekonečnými slovy.

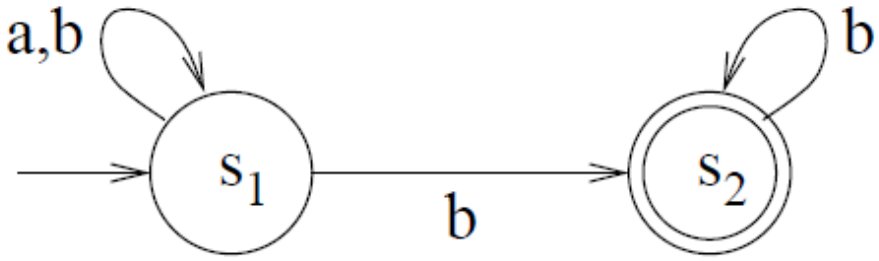
BA je pětice $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ kde:

- Q je konečná množina stavů
- Σ je vstupní abeceda,
- $\Delta \subseteq Q \times \Sigma \times Q$ je přechodová relace,
- $q_0 \in I$ je počáteční stav automatu,
- $F \subseteq Q$ je množina akceptujících stavů. Přijímá přesně ty běhy, v nichž alespoň jeden z nekonečně se vyskytujících stavů se nachází v F.

BA je nejjednodušším druhem ω-automatu. Zaujímají velmi specifické postavení v logice a teorii automatů. Našly uplatnění především v praktických aplikacích v lineární temporální logice a formální verifikaci systémů. Na Obr. 2.2 a Obr. 2.3 lze vidět deterministický a nedeterministický BA.



Obr. 2.2 - Deterministický Büchiho automat



Obr. 2.3 - Nedeterministický Büchiho automat

2.3.1 Zobecněné BA

V automatové teorii zobecněné BA (dále ZBA) je varianta BA. Akceptační podmínka je množina množin stavů F. Rozdíl mezi nimi je v akceptační podmínce. Běh je ZBA přijat, pokud navštíví alespoň jeden stav z každé sady akceptačních podmínek nekonečně často. ZBA mají stejnou vyjadřovací sílu jako BA, lze je tedy převést na ekvivalentní BA.

ZBA je pětice $\mathcal{A} = (Q, \Sigma, \Delta, I, \{F_1, \ldots, \mathrm{F}_n\})$ kde:

- Q je konečná množina stavů $\mathcal{A}$,
- Σ je vstupní abeceda,
- Δ: $Q \times \Sigma \times Q$ je přechodová funkce $\mathcal{A}$,
- $I \subseteq Q$ jsou počáteční stavy $\mathcal{A}$,
- $\{F_1, \ldots, \mathrm{F}_n\}$ je akceptační podmínka, která pro každé $1 \leq i \leq n, F_i \subseteq Q$.

2.3.2 Rabinovy automaty

Rabinův automat je druhem ω-automatu. Pro přijetí využívá následující podmínky pro určitou množinu Ω párů (E_i, F_i):
Rabinův automat $\mathcal{A}$ akceptuje přesně ty běhy ρ, pro které existuje pár (E_i, F_i) v Ω, že platí $E_i \cap Inf(\rho) = \emptyset$ a zároveň $F_i \cap Inf(\rho) \neq \emptyset$.

2.3.3 Streettovy automaty

Streettův automat je druhem ω-automatu. Jeho akceptační podmínka je množina dvojice množin. Pro přijetí využívá následující podmínky pro určitou množinu Ω párů (E_i, F_i):
Streettův automat $\mathcal{A}$ akceptuje přesně ty běhy ρ, pro které existuje pár (E_i, F_i) v Ω, že platí $E_i \cap Inf(\rho) \neq \emptyset$ a zároveň $F_i \cap Inf(\rho) = \emptyset$.

Podmínka Streettova automatu je tedy negací Rabinovi podmínky. Platí tedy, že deterministický Streett automat akceptuje přesně doplněk jazyka přijatého deterministickým Rabinovým automatem.

2.3.4 Paritní automaty

Paritní automat je automat, jehož množina stavů je $Q = (0,1,2,..,k)$ kde k je přirozené číslo. Akceptační podmínka paritního automatu je následující:
Přijímá běh ρ právě tehdy, když je nejmenší číslo v $Inf(\rho)$ lichý.

2.3.5 Mullerovi automaty

V teorii automatů je Mullerův automat (dále MA) druhem ω-automatu. Od ostatních ω-automatů se odlišuje podmínkou přijetí běhu. MA jsou definovány pomocí Mullerovi akceptační podmínky, jedná se o množinu všech nekonečně navštívených stavů, které musí patřit do množiny přijatých stavů.

Deterministický MA je pětice $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ kde:

- Q je konečná množina stavů $\mathcal{A}$,
- Σ je vstupní abeceda,
- $\delta: Q \times \Sigma \rightarrow Q$ je přechodová funkce $\mathcal{A}$,
- $q_0 \in Q$ je počáteční stav automatu $\mathcal{A}$,
- F je množina množin stavů. Formálně $F \subseteq 2^Q$, kde 2^Q je potenční množina z Q

Přijímá tedy přesně ty běhy, které se vyskytují nekonečně často v sadě stavů, z nichž některý patří do F.

2.4 ω-regulární jazyky

ω-regulární jazyky jsou třídy ω-jazyků, které zobecňují definice regulárních jazyků nad nekonečná slova.
ω-jazyk L je ω-regulární, pokud je ve formě:

- A^ω, kde A je neprázdný regulární jazyk neobsahující prázdný řetězec,
- AB, zřetězení regulárního jazyka A a ω-regulárního jazyka B,
- $A \cup B$, kde A a B jsou ω-regulární jazyky (toto pravidlo lze uplatnit pouze konečně mnohokrát)

Lze dokázat, že třída ω-regulárních jazyků je rovna třídě jazyků přijímajících nedeterministickými BA. Každý ω-regulární jazyk je přijat pomocí nedeterministického BA. Pomocí uzávěrových vlastností BA a strukturální indukce nad definicí ω-regulárních jazyků, může být snadno dokázáno, že můžeme pro jakýkoli ω-regulární jazyk sestrojit BA.

2.4.1 Uzávěrové vlastnosti ω-regulárních jazyků

Jak již bylo zmíněno výše, ω-regulární jazyky jsou rozpoznatelné pomocí BA. Lze tedy pomocí nich dokázat několik uzávěrových vlastností těchto jazyků. Z hlediska této práce jsou důležité vlastnosti sjednocení a průniku.

Uvažujme ω-regulární jazyk $\mathcal{L}(\mathcal{A})$ je rozpoznatelný BA $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \Delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}})$ a ω-regulární jazyk $\mathcal{L}(\mathcal{B})$ je rozpoznatelný BA $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \Delta_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}})$. Pak:

- $\mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{B})$ je ω-regulární jazyk. Dále platí že $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$. Důkaz: Sestrojme BA $\mathcal{C} = (Q_{\mathcal{C}}, \Sigma, \Delta_{\mathcal{C}}, I_{\mathcal{C}}, F_{\mathcal{C}})$, kde:
  - $Q_{\mathcal{C}} = Q_{\mathcal{A}} \cup Q_{\mathcal{B}}$,
  - $\Delta_{\mathcal{C}} = \Delta_{\mathcal{A}} \cup \Delta_{\mathcal{B}}$,
  - $I_{\mathcal{C}} = I_{\mathcal{A}} \cup I_{\mathcal{B}}$,
  - $F_{\mathcal{C}} = F_{\mathcal{A}} \cup F_{\mathcal{B}}$.

  Sjednocení BA jazyků je podobné jako u nedeterministických konečných automatů.

- $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$ je ω-regulární jazyk. Dále platí že $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$. Důkaz: Sestrojme BA $\mathcal{C} = (Q_{\mathcal{C}}, \Sigma, \Delta_{\mathcal{C}}, I_{\mathcal{C}}, F_{\mathcal{C}})$, kde:
  - $Q_{\mathcal{C}} = Q_{\mathcal{A}} \times Q_{\mathcal{B}} \times \{1,2\}$,
  - $\Delta_{\mathcal{C}} = \Delta_1 \cup \Delta_2$, kde:
    - $\Delta_1 = \{((q_{\mathcal{A}}, q_{\mathcal{B}}, 1), a, (q'_{\mathcal{A}}, q'_{\mathcal{B}}, i)) \mid (q_{\mathcal{A}}, a, q'_{\mathcal{A}}) \in \Delta_{\mathcal{A}}$ a $(q_{\mathcal{B}}, a, q'_{\mathcal{B}}) \in \Delta_{\mathcal{B}}$ a pokud $q_{\mathcal{A}} \in F_{\mathcal{A}}$, tak $i = 2$ jinak $i = 1\}$,
    - $\Delta_2 = \{((q_{\mathcal{A}}, q_{\mathcal{B}}, 2), a, (q'_{\mathcal{A}}, q'_{\mathcal{B}}, i)) \mid (q_{\mathcal{A}}, a, q'_{\mathcal{A}}) \in \Delta_{\mathcal{A}}$ a $(q_{\mathcal{B}}, a, q'_{\mathcal{B}}) \in \Delta_{\mathcal{B}}$ a pokud $q_{\mathcal{B}} \in F_{\mathcal{B}}$, tak $i = 1$ jinak $i = 2\}$,
  - $I_{\mathcal{C}} = I_{\mathcal{A}} \times I_{\mathcal{B}} \times \{1\}$,
  - $F_{\mathcal{C}} = \{(q_{\mathcal{A}}, q_{\mathcal{B}}, 2) \mid q_{\mathcal{B}} \in F_{\mathcal{B}}\}$.

3 Existující knihovny pro práci s BA a knihovna VATA

Cílem této práce je rozšířit již existující knihovnu VATA o podporu práce s Büchiho automaty. V této kapitole se nachází popis knihovny VATA, taktéž je zde stručný popis již existujících knihoven pro práci s Büchiho automaty.

3.1 Existující knihovny pro práci s BA

Büchiho automaty jsou rozšířené a používané v teoretické informatice a také v model checkingu, existuje proto několik knihoven pro práci s těmito automaty. Konkrétně se zde nachází stručný popis knihovny SPOT a grafického nástroje GOAL.

3.1.1 Knihovna SPOT

SPOT [4] je knihovna napsána v jazyce C++ a nabízí model checking, který lze kombinovat a propojit s nástroji třetích stran k vytvoření testovaného modelu. Knihovna SPOT je založena na převodu na zobecněné BA, které není třeba převádět na BA pro testování inkluze. Knihovna podporuje on-the-fly výpočty. Může být dále rozšiřována a využívána v jiných nástrojích.

3.1.2 Nástroj GOAL

Nástroj GOAL [5] je grafický interaktivní nástroj pro práci s BA a formulemi temporální logiky. Z části také podporuje i další varianty ω-automatů. Tento nástroj slouží především ke vzdělávacím účelům a to zejména k lepšímu pochopení toho, jak fungují BA, a jak jsou spojeny s lineární temporální logikou. První verze nástroje byla vydána 1. 6. 2007. Díky tomu, že umožňuje přistupovat k jednotlivým funkcím prostřednictvím programů a skriptů, je GOAL vhodný pro další podpůrné výzkumy.

Zkratka GOAL byla původně odvozena z „Graphical Tool for Omega-Automata and Logics“ a později se přeformulovala jako „Games, Omega-Automata, and Logics“, jak byla postupně přidávána podpora pro ω-regulární hry. Dlouhodobý cíl tvůrců tohoto nástroje je, aby nástroj uměl zvládnout všechny běžné varianty ω-automatů a logiky, která je ekvivalentem těchto automatů. Nástroj se neustále vylepšuje a rozšiřuje v pravidelných intervalech. Nástroj GOAL je stažitelný z [13].

3.1.3 Nástroj RABIT

Nástroj RABIT [6] slouží pro testování jazykové inkluze mezi dvěma Büchiho automaty. Využívá k tomu přístup založený na Ramseyho větě. Dále podporuje kontrolu jazykové ekvivalence a jazykovou universalitu BA. Vzniklo několik verzí tohoto nástroje. V nejnovější verzi se nachází také minimalizace BA, která zvyšuje efektivitu operace testování jazykové inkluze.

RABIT byl rozšířen o podporu práce s konečnými automaty. Pro ně lze taktéž provádět testování jazykové inkluze, ekvivalence a jazykové universality.

Vstupní formát pro oba automaty je podporován také nástrojem GOAL, který byl popsán v 3.1.2.

3.2 Knihovna VATA

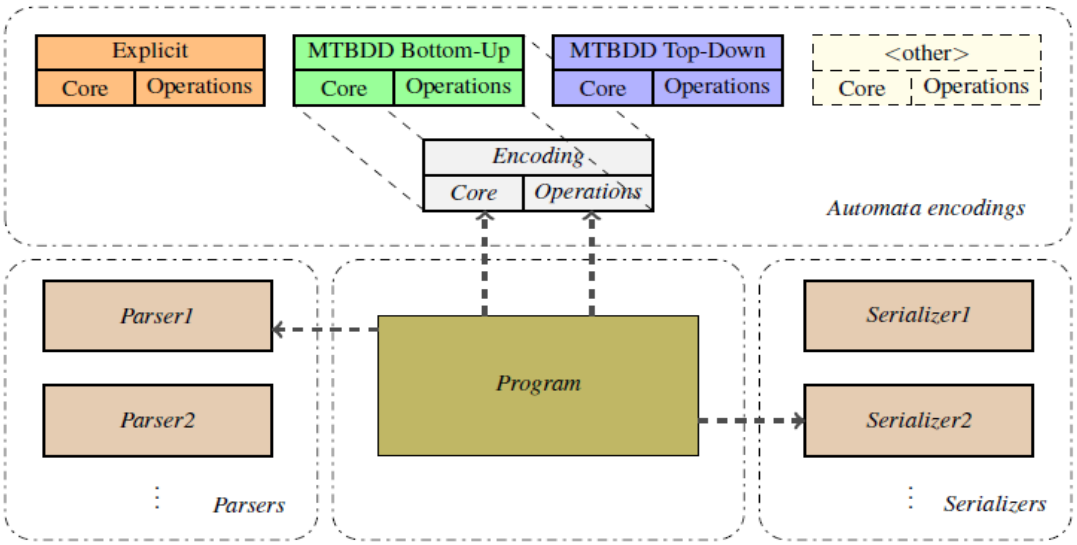
VATA [3] je vysoce optimalizovaná knihovna pro práci s nedeterministickými stromovými konečnými automaty (stažitelná z [12]). Hlavním cílem knihovny je její využití v oblasti formální verifikace, ale může být také efektivně využita v jiných oblastech. Knihovna je implementována v jazyce C++ s využitím knihovny Boost C++.

Stromové automaty rozšiřují abecedu o ohodnocovací funkci $\#: \Sigma \to \mathbb{N}$. Stromový automat (SA) je čtveřice $\mathcal{A} = (Q, \Sigma, \Delta, F)$, kde Q je konečná množina stavů, Σ je řídicí abeceda, $F \subseteq Q$ je konečná množina koncových stavů a Δ je konečná množina přechodů. Každý přechod je následujícího formátu$\big((q_1, \dots, q_n), a, q\big)$ kde $q, q_1, \dots, q_n \in Q$, $a \in \Sigma$ kde $\#(a) = n$. $\big((q_1, \dots, q_n), a, q\big)$ může být označeno jako $(q_1, \dots, q_n) \xrightarrow{a} q$ (reprezentace SA zdola-nahoru) nebo jako $q \xrightarrow{a} (q_1, \dots, q_n)$, (reprezentace SA shora-dolů). Obě tyto notace jsou ekvivalentní. Speciálním případem jsou přechody kde $n = 0$, které se nazývají *listové* přechody.

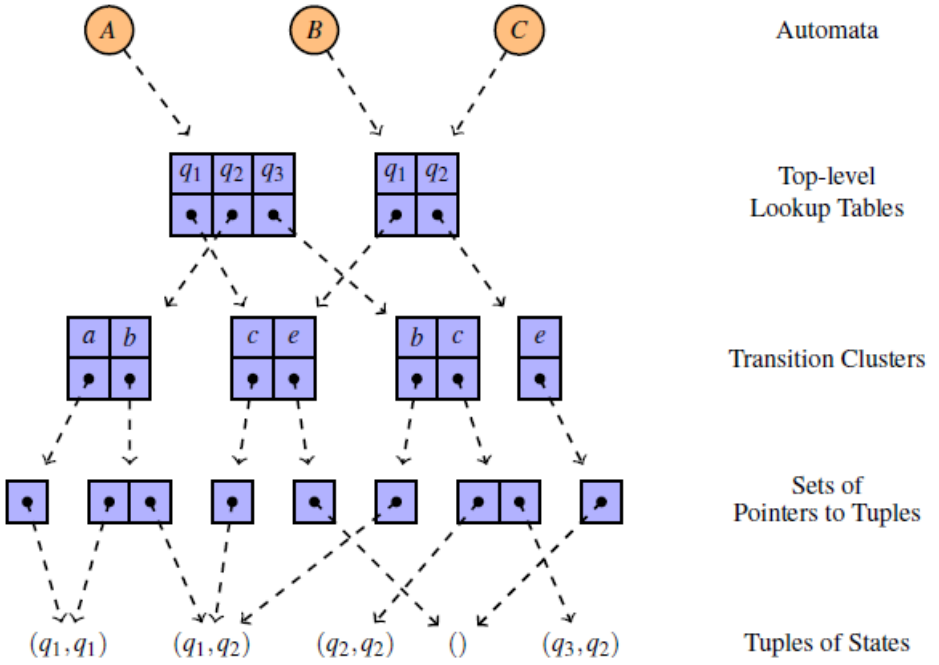
3.2.1 Návrh knihovny

Knihovna obsahuje dvě podporované reprezentace stromových automatů a to explicitní a semi-symbolické. Hlavním rozdílem mezi těmito reprezentacemi je v datových strukturách pro ukládání přechodu stromového automatu. Semi-symbolická reprezentace používá multi-terminální binární rozhodovací diagramy pro ukládání přechodové tabulky automatu. Je určeno především pro automaty s velkými abecedami.

Knihovna VATA je napsána modulární cestou, lze ji tedy snadno rozšířit o další moduly. Díky modularitě může jakákoli nově přidaná reprezentace sdílet ostatní části knihovny, jakou jsou parsery a serializéry. Hlavní koncept knihovny je popsán na Obr. 3.1.



Obr. 3.1- Návrh knihovny VATA. Obrázek převzat z [3].



Obr. 3.2 - Příklad explicitního kódování přechodových realcí stromových automatů $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$. Zejména lze vidět, že $\mathcal{A}$ obsahuje přechod $q_1 \xrightarrow{c} (q_1, q_2)$: stačí sledovat odpovídající ukazatele. Kromě toho $\mathcal{B}$ také obsahuje stejný přechod (a odpovídající část sdílí s $\mathcal{A}$). Na závěr $\mathcal{C}$ obsahuje stejné přechody jako $\mathcal{B}$. Obrázek převzat z [3].

3.2.2 Explicitní kódování

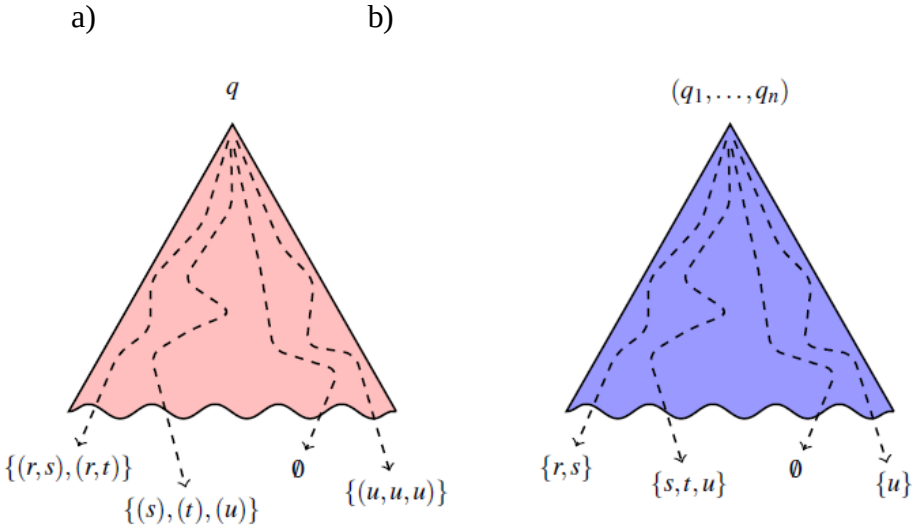
V explicitní reprezentaci SA použité v knihovně VATA mají přechody tvar $q \xrightarrow{a} (q_1, \ldots, q_n)$ a data jsou uchována v hierarchické struktuře podobné hashovací tabulce. Vrchní částí této datové struktury je hashovací tabulka, která mapuje stavy do přechodových klusterů. Tyto

klustery jsou také vyhledávací tabulky a mapují symboly vstupní abecedy na skupinu ukazatelů na n-tice stavů. Uložení n-tic stavů může být velmi paměťově náročné, proto je každá n-tice uložená v paměti pouze jednou, i když je odkazována různými přechody. Vkládání nového přechodu do této struktury vyžaduje amortizovaně konstantní počet kroků. Tato struktura je znázorněna na Obr. 3.2.

3.2.3 Semi-symbolické kódování

Semi-symbolická reprezentace využívá multi-terminální binární rozhodovací diagramy (MTBDDs) k reprezentaci přechodových funkcí stromového automatu. MTBDDs jsou rozšíření binárních rozhodovacích diagramů (BBDs), populární datové struktury pro kompaktní kódování a manipulování s výrokovými logickými formulemi.

Jsou zde podporovány dva druhy semi-symbolické reprezentace: shora-dolů a zdola-nahoru. Hlavní rozdíl mezi těmito dvěma reprezentacemi je v ukládání symbolů v MTBDD. V reprezentaci zdola-nahoru jsou vstupní symboly ukládány s jejich aritou, aby bylo možno rozlišit mezi dvěma instancemi se stejnými symboly s rozdílnou aritou. Reprezentace shora-dolů nepotřebuje ukládat aritu symbolů, protože ji lze získat z n-tic na levé straně přechodu. Příklad semi-symbolického kódování lze vidět na Obr. 3.3.



Obr. 3.3 - Reprezentace (a) shora-dolů a (b) zdola-nahoru semi-symbolické reprezentace přechodových funkcí. Cesty v MTBDD se shodují se symboly. Obrázek převzat z [3].

3.2.4 Operace

V knihovně VATA si uživatel může vybrat ze tří reprezentací: explicitní shora-dolů, semi-symbolické shora-dolů a semi-symbolické zdola-nahoru. V závislosti na výběru této reprezentace jsou dostupné určité operace nad stromovým automatem. Následující operace jsou podporovány alespoň jedním z těchto reprezentací: sjednocení, průnik, eliminace nedostupných stavů (shora-dolů, zdola-nahoru), kontrola jazykové inkluze (shora-dolů, zdola-nahoru), výpočet relace simulace (nahoru, dolů) a redukce velikosti na základě relace simulační ekvivalence. V některých případech je dostupno více implementací jedné operace,

jako například jazykové inkluze. Je to způsobeno tím, že rozdílné implementace jsou založeny na rozdílných heuristikách a mohou tak fungovat lépe pro různé aplikace. Operace, které jsou podporovány u jednotlivých reprezentací, lze vidět na Obr. 3.4.

Operace	Explicitní kódování	Semi-symbolické kódování	
	shora - dolů	shora-dolů	zdola-nahoru
Sjednocení	A	A	A
Průnik	A	A	A
Komplementace	A	N	N
Odstranění nedostupných stavů	A	A	A
Odstranění neukončujících stavů	A	A	A
Simulace směrem vzhůru	A	A	A
Simulace směrem dolů	A	A	N
Testování inkluze zdola-nahoru	A	N	A

Obr. 3.4 - Tabulka implementovaných operací u jednotlivých kódování v knihovně VATA.

3.2.5 Rozšíření pro Büchiho automaty

Knihovna VATA je psána modulární cestou, díky čemuž je snadno rozšiřitelná o další modul. Je tedy vhodné nevytvářet úplně novou knihovnu pro práci s BA, ale implementovat rozšíření knihovny VATA v jazyce C++.

Hlavním cílem je poskytnout efektivní implementace operací pro práci nad Büchiho automaty, především algoritmus pro testování jazykové inkluze. Existuje několik přístupů a to například Ramsey-based nebo Rank-based [7]. Tato práce je zaměřena na první z nich. Knihovna již obsahuje několik implementací pro práci s několika druhy automatů a to konečnými a stromovými automaty. Pro práci s BA v knihovně VATA budou využity především datové struktury, které jsou podobné datovým strukturám pro práci s konečnými automaty. Nebude zde implementován pouze algoritmus pro jazykovou inkluzi, ale také základní operace jako jsou sjednocení, průnik, redukce počtu stavů a také základní operace pro manipulaci s BA (např. přidání přechodu, označení stavu za počáteční atd.). Nové rozšíření bude podporovat pouze explicitní kódování a bude využívat již některé existující prvky knihovny VATA. Jedná se o serializéry, parsery nebo také výpočet relace simulace nad stavy automatu.

4 Jazyková inkluze

Testování jazykové inkluze mezi dvěma konečně stavovými automaty je jeden z hlavních problémů v teorii automatů. Má mnoho praktických využití. Pokud uvedeme příklad, tak při model checkingu můžeme mít systém, kdy jeden konečně stavový automat udává jeho chování a druhý udává specifikaci tohoto systému. Testování, zda chování systému vyhovuje jeho specifikaci, se zde redukuje na problém jazykové inkluze.

Testování jazykové inkluze je obecně problém mezi dvěma automaty. Tato práce se zabývá Büchiho automaty, proto testování jazykové inkluze bude prováděno na nich. Testování jazykové inkluze mezi dvěma nedeterministickými Büchiho automaty je velmi výpočetně náročná operace (jedná se o tzv. PSPACE-úplný problém) a bylo tedy věnováno mnoho úsilí na vypracování přístupů, které dokáží efektivně řešit praktické případy. Problém jazykové inkluze mezi dvěma BA $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \Delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}})$ a $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \Delta_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}})$ je testování zda $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$. Naivní přístup testování jazykové inkluze vypadá následovně. Nejprve se vytvoří komplement k automatu $\mathcal{B}$, automat $\mathcal{B}^C$, a poté se provede testování prázdnosti průniku $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}^C)$, platí že $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B}) <=> \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}^C) = \emptyset$. Zde je hlavní problém, že $\mathcal{B}^C$ je až exponenciálně větší než $\mathcal{B}$. Proto se pro praktická řešení problému jazykové inkluze musejí využít jiné přístupy. V této práci se zaměříme na optimalizovaný přístup založený na Ramseyho větě (Ramsey-based).

4.1 Ramsey-based

Jádrem přístupu založeného na Ramseyho větě (Ramsey-based) [8,9] pro jazykovou inkluzi mezi dvěma Büchiho automaty $\mathcal{A}$ a $\mathcal{B}$ je tzv. *supergraf*. Jedná se o datovou strukturu reprezentující třídu konečných slov sdílejících podobné chování v obou automatech. Ramsey-based algoritmus zahrnuje tři fáze. Nejprve se jedná o fázi inicializace, kde se identifikuje množina počátečních supergrafů. V následující fázi se hledají tzv. lasa v supergrafech, která jsou supergrafy iterativně generována kompozicí s počátečními supergrafy. V poslední fázi testuje, zda některé páry supergrafů obsahují protipříklad. Supergrafy jsou samy o sobě malé, třetí fázi lze tedy provádět efektivně, avšak základní verze algoritmu leží v až exponenciálním počtu vygenerovaných supergrafů. Z tohoto důvodu je základním úkolem Ramsey-based algoritmu omezit počet supergrafů. Toho může být dosaženo například odstraněním některých supergrafů nebo také existuje několik metod, kde jsou tyto supergrafy minimalizovány.

V této práci byl zvolen podobný přístup, jako je popsán v [8]. Z dokumentu lze vidět, že testování jazykové inkluze je velmi podobné jako testování universality. V následující sekci definujeme některé důležité pojmy z hlediska testování jazykové inkluze.

4.1.1 Grafy, hrany a simulace

Následující definice jsou důležité při Ramsey-based testování universality BA. V této sekci budeme uvažovat BA $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$.

Nechť $E = Q \times \{-1, 0, 1\} \times Q$ a nechť $G_{\mathcal{A}}$ je největší podmnožina 2^E jejíž prvky obsahují přesně jeden člen z $\{< p, -1, q >, < p, 0, q >, < p, 1, q >\}$ pro každý $p, q \in Q$. Každý prvek v $G_{\mathcal{A}}$ je $\{-1, 0, 1\}$-hranově ohodnocený graf nad množinou vrcholů Q.

Pro každý pár stavů $p, q \in Q$, definujeme následující tři jazyky:
(1) $\mathcal{L}(p, 1, q) = \{w \in \Sigma^+ \mid p \overset{w}{\Rightarrow}_F q\}$,
(2) $\mathcal{L}(p, 0, q) = \{w \in \Sigma^+ \mid p \overset{w}{\Rightarrow} q \wedge \neg(p \overset{w}{\Rightarrow}_F q)\}$,
(3) $\mathcal{L}(p, -1, q) = \{w \in \Sigma^+ \mid \neg(p \overset{w}{\Rightarrow} q)\}$.

Jazyk grafu $g \in G_{\mathcal{A}}$ je definován jako průnik jazyků jednotlivých hran v g, $\mathcal{L}(g) = \cap_{<p,a,q> \in g} \mathcal{L}(p, a, q)$. Pro každé slovo $w \in \Sigma^+$ a každý pár $p, q \in Q$ existuje přesně jedna hrana $< p, a, q >$ taková, že $w \in \mathcal{L}(p, a, q)$. Proto jazyky grafů v $G_{\mathcal{A}}$ tvoří část Σ^+, protože se jedná o průniky jazyků hran. Y_{gh} značí ω-regulární jazyk $\mathcal{L}(g).\mathcal{L}(h)^{\omega}$.

Množina $G_{\mathcal{A}}^f = \{g \in G_{\mathcal{A}} \mid \mathcal{L}(g) \neq \emptyset\}$ může být iterativně generována následovně. Nejprve pro dané grafy $g, h \in G_{\mathcal{A}}$, je jejich kompozice $g; h$ definována jako:
$\{< p, -1, q > | \forall t \in Q: (< p, a, t > \in g \wedge < t, b, q > \in h) \rightarrow (a = -1 \vee b = -1)\} \cup$
$\{< p, 0, q > | \exists r \in Q: < p, 0, r > \in g \wedge < r, 0, q > \in h \wedge$
$\wedge \forall t \in Q: (< p, a, t > \in g \wedge < t, b, q > \in h) \rightarrow (a \neq 1 \vee b \neq 1)\} \cup$
$\{< p, 1, q > | \exists r \in Q: < p, a, r > \in g \wedge < r, b, q > \in h \wedge (a = 1 \vee b = 1)\}$.

Pro všechny $a \in \Sigma$, definujeme graf jednoho symbolu $g_a = \{< p, -1, q > \mid q \notin \Delta(p, a)\} \cup \{< p, 0, q > \mid p \in (Q \backslash F) \wedge q \in (\Delta(p, a) \backslash F)\} \cup \{< p, 1, q > | q \in \Delta(p, a) \wedge \{p, q\} \cap F \neq \emptyset\}$. Nechť $G_{\mathcal{A}}^1 = \{g_a \mid a \in \Sigma\}$. Opakovanou kompozicí grafů v $G_{\mathcal{A}}^1$ lze získat množinu $G_{\mathcal{A}}^f$. Graf g je členem $G_{\mathcal{A}}^f$, $g \in G_{\mathcal{A}}^f$ tehdy a jen tehdy, když $\exists g_1, \ldots, g_n \in G_{\mathcal{A}}^1 : g = g_1; \ldots; g_n$.

Laso hledající test (zkráceně LFT) páru grafů $< g, h >$, kde $g, h \in G_{\mathcal{A}}^f$, kontroluje existenci tzv. *lasa*, kdy se hledá cesta v g ze stavu $p \in I$ do stavu $q \in F$ a cesta v h z q zpět do q. Máme pár grafů $< g, h >$, které projdou laso hledajícím testem (zkráceně $LFT(g, h)$) tehdy a jen tehdy, když je $< p, a_0, q_0 > \in g$ a existuje nekonečná posloupnost hran $< q_0, a_1, q_1 >, < q_1, a_2, q_2 >, \ldots \in h$, kde $p \in I, a_i \in \{0,1\}$ pro všechny $i \geq 0$ a $a_j = 1$ pro nekonečně mnoho $j \in \mathbb{N}$.

V algoritmu, založeném na Ramseyho větě, se také pracuje s relací pokrytí, která se značí $\sqsubseteq$ a je definována následovně. Pro grafy $f, g, h \in G_{\mathcal{A}}$, říkáme, že $g \sqsubseteq h$ tehdy a jen

tehdy, pokud pro každou hranu $< p, a, q > \in g$, existuje hrana $< p, a', q > \in h$ taková, že platí $a \leq a'$.

Simulace je relace $R \subseteq Q \times Q$ taková že pokud pRr, pak $p \in F \Longrightarrow r \in F$ a pro každý přechod $p \xrightarrow{a} p'$, existuje přechod $r \xrightarrow{a} r'$ takový, že $p'Rr'$. Pro BA $\mathcal{A}$ existuje unikátní maximální simulace, která je kvaziuspořádání (reflexivní a tranzitivní relace), nazývá se *simulační kvaziuspořádání* a označuje se $\preccurlyeq_{\mathcal{A}}$ nebo pouze $\preccurlyeq$, když $\mathcal{A}$ je zřejmé z kontextu.

Pro zrychlení Ramsey-based metody z hlediska použití při testování universality automatu se využívá dvou optimalizací.

První z nich je založena na poznatku, že relace pokrytí $\sqsubseteq$ může být zobecněna využitím simulačního kvaziuspořádání. Tento pojem je nazýván relace založená na simulačním pokrytí a značí se $\sqsubseteq^{\forall\exists}$. V $\sqsubseteq$ jsou vyžadovány hrany ve formátu $< p, a, q >$, které mohou být pokryty pouze hranami ve formátu $< p, a', q' >$ s $a \leq a' \wedge q' = q$, poslední rovnost se dá zobecnit za pomocí relace simulace. To popisuje následující definice $\sqsubseteq^{\forall\exists}$. Pro každý graf $g, h \in G_{\mathcal{A}}$ platí, že $g \sqsubseteq^{\forall\exists} h$ tehdy a jen tehdy, když pro každou hranu $< p, a, q > \in g$ existuje hrana $< p, a', q' > \in h$ taková, že $a \leq a'$ a $q \preccurlyeq q'$.

Druhá optimalizace je založena na zjednodušování struktur jednotlivých grafů v $G_{\mathcal{A}}$ za pomocí simulačního pokrytí. Díky tomuto je umožněno „zeslabit“ symboly na některých hranách v grafech. Máme-li množinu symbolů ohodnocující grafy $\{1,0,-1\}$, tak můžeme některé symboly $\{0,1\}$ nahradit symbolem -1. Tak je snížena složitost operací, jako je operace hledání lasa (LFT), kontrola pokrytí nebo také grafová kompozice. Časová složitost těchto operací je polynomiální vzhledem k počtu hran se symboly z množiny $\{0,1\}$, a tak menší grafy snižují jejich složitost. Za účelem redukování grafů je třeba definovat operaci Min, která mapuje každý graf z $f \in G_{\mathcal{A}}$ na graf $Min(f) \in G_{\mathcal{A}}$ takový, že $Min(f) \lesssim^* f$. $\lesssim^*$ zde znamená, že $Min(f)$ je shodný s f nebo se jedná o zredukovanou verzi f. Tuto redukovanou verzi můžeme odvodit z f zjednodušením některých hran, které jsou začleněny ostatními hranami v obou grafech $Min(f)$ i f. Dále definujeme $\lesssim^*$. Pro libovolné grafy $g, h \in G_{\mathcal{A}}$ zapisujeme $g \lesssim h$ tehdy a pouze tehdy, když existuje hrana $< p, a, q > \in h$ a $< p, a', q' > \in g \cap h$, kde $a \leq a'$, $q \preccurlyeq q'$, a $g = (h \setminus \{< p, a, q >\}) \cup \{< p, a'', q >\}$, kde $a'' \leq a$. Relace $\lesssim^*$ je transitivně reflexivní uzávěr relace $\preccurlyeq$. Následující definice popisuje množinu redukovaných grafů se neprázdným jazykem. Metodu Min můžeme v praxi implementovat tak, že je navrácen graf, který je nejmenší vzhledem k $\lesssim^*$. To se provede snížením hodnot symbolů u co největšího počtu hran na -1.

4.1.2 Supergrafy

Předešlá část se používá především pro testování universality Büchiho automatu, a tak musíme pro jazykovou inkluzi tyto definice upravit. Nechť $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}})$ a $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \delta_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}})$ jsou dva Büchiho automaty. Dále pak mějme relace $\preccurlyeq_{\mathcal{A}}$ a $\preccurlyeq_{\mathcal{B}}$ maximálních simulací na $\mathcal{A}$ a $\mathcal{B}$. Definujme množinu $E_{\mathcal{A}} = Q_{\mathcal{A}} \times Q_{\mathcal{A}}$. Každý prvek množiny $E_{\mathcal{A}}$ je hrana $< p, q >$ taková, že existuje cesta z p do q v $\mathcal{A}$. Definujme jazyk hrany $< p, q >$ jako $\mathcal{L}(p, q) = \{w \in \Sigma^+ \mid p \overset{w}{\Rightarrow} q\}$.

Nechť $S_{\mathcal{A},\mathcal{B}} = E_{\mathcal{A}} \times G_{\mathcal{B}}$. Prvek z množiny $S_{\mathcal{A},\mathcal{B}}$ se nazývá *supergraf* $\boldsymbol{g} = <\bar{g}, g>$. Pro každý supergraf $\boldsymbol{g} \in S_{\mathcal{A},\mathcal{B}}$, je jazyk $\mathcal{L}(\boldsymbol{g})$ definovaný jako $\mathcal{L}(\bar{g}) \cap L(g)$. $Z_{\boldsymbol{gh}}$ značí ω-regulární jazyk $\mathcal{L}(\boldsymbol{g}).\mathcal{L}(\boldsymbol{h})^{\omega}$. Můžeme říct $Z_{\boldsymbol{gh}}$ je správně kompozitní, když $\bar{g} = <p, q>$ a $\bar{h} = <q, q>$ pro nějaké $p \in I_{\mathcal{A}}$ a $q \in F_{\mathcal{A}}$. Lze si všimnout, že podle definice kompozice je každá správně kompozitní množina $Z_{\boldsymbol{gh}}$ obsažena v $\mathcal{L}(\mathcal{A})$. Supergrafy v $S^{f}_{\mathcal{A},\mathcal{B}} = \{\boldsymbol{g} \in S_{\mathcal{A},\mathcal{B}} \mid \mathcal{L}(\boldsymbol{g}) \neq \emptyset\}$ (tedy supergrafy s neprázdnými jazyky) mohou být vytvořeny následujícím způsobem. Nejprve, supergrafy $\boldsymbol{g} = <<p_{\boldsymbol{g}}, q_{\boldsymbol{g}}>, g>$ a $\boldsymbol{h} = <<p_{\boldsymbol{h}}, q_{\boldsymbol{h}}>, h>$ v $S_{\mathcal{A},\mathcal{B}}$ jsou kompozitní pouze a jen tehdy, pokud $q_{\boldsymbol{g}} = p_{\boldsymbol{h}}$. Pro všechny symboly $a \in \Sigma$, definujeme množinu supergrafů $S^{a} = \{<<p, q>, g_a> \mid q \in \delta_{\mathcal{A}}(p, a)\}$. Nechť $S^{1}_{\mathcal{A},\mathcal{B}} := \cup_{a \in \Sigma} S^{a}$. Opakovaným skládáním $S^{1}_{\mathcal{A},\mathcal{B}}$ získáme množinu $S^{f}_{\mathcal{A},\mathcal{B}}$. Dvojice supergrafů $<\boldsymbol{g} = <\bar{g}, g>, \boldsymbol{h} = <\bar{h}, h>>$ projde testem dvojice grafů (zkráceně se označuje jako $DGT(\boldsymbol{g}, \boldsymbol{h})$) tehdy a jen tehdy, když $Z_{\boldsymbol{gh}}$ není správně kompozitní nebo $LFT(g, h)$.

Vstup: *BA* $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \Delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}}), \mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \Delta_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}})$ *a množina* $S^{1}_{\mathcal{A},\mathcal{B}}$
Výstup: *TRUE pokud platí, že* $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$. *Jinak FALSE.*
$Next := \{Min_S(\boldsymbol{g}) | \boldsymbol{g} \in S^{1}_{\mathcal{A},\mathcal{B}}\}$; $Init := \emptyset$;
while $(Next \neq \emptyset)$ ***do*** {
  Vem a odstraň supergaf $\boldsymbol{g}$ *z Next*;
  if $(\exists f \in Init : \boldsymbol{f} \sqsubseteq_S^{\forall\exists} \boldsymbol{g})$ ***then*** *Vymaž* $\boldsymbol{g}$ *a pokračuj*;
  Odstraň všechny supergrafy $\boldsymbol{f}$ *z množiny Init*;
  Vlož $\boldsymbol{g}$ *do Init*;
}
$Next := Init; Processed = \emptyset$;
while $(Next \neq \emptyset)$ ***do*** {
  Vem supergraf $\boldsymbol{g}$ *z Next*;
  if $(\neg RDGT(\boldsymbol{g}, \boldsymbol{g}) \vee \exists \boldsymbol{h} \in Processed : \neg RDGT(\boldsymbol{g}, \boldsymbol{h}) \vee \neg RDGT(\boldsymbol{h}, \boldsymbol{g}))$ ***then***
    Vrať FALSE;
  Odstraň z Next a vlož jej do Processed;
  foreach$(\boldsymbol{h} \in Init, kde <\boldsymbol{g}, \boldsymbol{h}> jsou\ kompozitní)$ ***do*** {
    $\boldsymbol{f} := Min_S(\boldsymbol{g}; \boldsymbol{h})$;
    if $(\exists \boldsymbol{k} \in Processed \cup Next : \boldsymbol{k} \sqsubseteq_S^{\forall\exists} \boldsymbol{f})$ ***then*** *Vymaž* $\boldsymbol{f}$ *a pokračuj*;
    Odstraň všechny supergrafy $\boldsymbol{k}$ *z* $Processed \cup Next$
    Vlož $\boldsymbol{f}$ *do Next*;
  }
}

Algoritmus 4.1 - Vylepšený algoritmus pro testování jazykové inkluze dvou BA pomocí přístupu založený na Ramseyho větě převzat z [8].

4.1.3 Vylepšení jazykové inkluze prostřednictvím simulace

V této části bude popsáno vylepšení jazykové inkluze prostřednictvím relace simulace stejně, jako je v [8]. Při vylepšení se také využívá simulační pokrytí tak, jak je popsána v sekci 4.1.1, avšak zde pracujeme se supergrafy a musíme definice poupravit.

Relace simulačního pokrytí $\sqsubseteq^{\forall\exists}$ vzhledem k supergrafům vypadá následovně: Nechť $\boldsymbol{g} = << p_{\boldsymbol{g}}, q_{\boldsymbol{g}} >, g >$ a $\boldsymbol{h} = << p_{\boldsymbol{h}}, q_{\boldsymbol{h}} >, h >$ jsou dva supergrafy v $S_{\mathcal{A},\mathcal{B}}$. Pak $\boldsymbol{g} \sqsubseteq_S^{\forall\exists} \boldsymbol{h}$ pouze a jen tehdy, když $p_{\boldsymbol{g}} = p_{\boldsymbol{h}}, q_{\boldsymbol{g}} \succcurlyeq_{\mathcal{A}} q_{\boldsymbol{h}}$ a $g \sqsubseteq^{\forall\exists} h$.

Jelikož chceme pracovat se supergrafy, které jsou minimální, píšeme $\sqsubseteq_S^{\forall\exists}$, potřebujeme změnit definici správnosti a testu dvojice grafů. Říkáme, že $Z_{\boldsymbol{gh}}$ je slabě správně kompozitní pouze a jen tehdy, když $\bar{g} = < p, q >$ a $\bar{h} = < q_1, q_2 >$, kde $p \in I_{\mathcal{A}}$, $q_2 \in F_{\mathcal{A}}$, $q \succcurlyeq_{\mathcal{A}} q_1$ a $q_2 \succcurlyeq_{\mathcal{A}} q_1$. Supergrafy $\boldsymbol{g}, \boldsymbol{h} \in S_{\mathcal{A},\mathcal{B}}$ projdou uvolněným testem dvojice grafů (anglicky relaxed double-graph test, zkráceně RDGT), který se zapisuje $RDGT(\boldsymbol{g}, \boldsymbol{h})$, pouze tehdy, když buď $Z_{\boldsymbol{gh}}$ není slabě správně kompozitní (zároveň každá slabě správně kompozitní množina $Z_{\boldsymbol{gh}}$ je obsažena v $\mathcal{L}(\mathcal{A})$) nebo $LFT(g, h)$.

Potřebujeme také upravit další části pro práci s grafy z předešlé sekce tak, aby mohli být použity pro supergrafy. Jedná se o funkce Min a $\lesssim^*$. Pro libovolné dva supergrafy $\boldsymbol{g} = < \bar{g}, g >$ a $\boldsymbol{h} = < \bar{h}, h >$ z množiny $S_{\mathcal{A},\mathcal{B}}$ zapisujeme $\boldsymbol{g} \lesssim_S^* \boldsymbol{h}$ tehdy a pouze tehdy když $\bar{g} = \bar{h}$ a $g \lesssim_S^* h$. Potom $S_{\mathcal{A},\mathcal{B}}^m = \{\boldsymbol{g} \in S_{\mathcal{A},\mathcal{B}} \mid \exists \boldsymbol{h} \in S_{\mathcal{A},\mathcal{B}}^f : \boldsymbol{g} \lesssim_S^* \boldsymbol{h}\}$. $Min_S(\boldsymbol{g})$ vypočítá graf, který je $\lesssim_S^*$-menší než $\boldsymbol{g}$. Je také možno mít tuto operaci nedeterministickou tak, že $Min_S(\bar{g}, g) = < \bar{g}, Min_S(g) >$.

Na Algoritmu 4.1 je zobrazen kompletní vylepšený algoritmus Ramsey-based pro jazykovou inkluzi dvou BA.

4.1.4 Popis algoritmu

V této části se nachází stručný popis k Algoritmu 4.1, který zobrazuje optimalizovaný algoritmus Ramsey-based pro testování jazykové inkluze. Vstupem jsou dva BA $\mathcal{A}$, $\mathcal{B}$ a množina supergrafů $S_{\mathcal{A},\mathcal{B}}^1$, která je z těchto automatů vygenerována. Výstupem pak je $TRUE$ nebo $FALSE$ v závislosti na tom, zda byla prokázána jazyková inkluze dvou vstupních automatů nebo nikoliv. Před startem algoritmu se vygenerovaná množina $S_{\mathcal{A},\mathcal{B}}^1$ minimalizuje pomocí funkce Min_S. V algoritmu se pracuje s několika pomocnými množinami pro ukládání supergrafů jako jsou $Next, Init$ a $Processed$.

Algoritmus lze rozdělit do dvou částí. První z nich je příprava před hlavní částí, kdy se supergrafy mezi sebou testují, zda se navzájem pokrývají. Všechny supergrafy, které jsou pokryty jinými, tak jsou odstraněny a nadále se s nimi v algoritmu nepracuje. Poté, co jsou tyto pokryté supergrafy odstraněny, se přechází na hlavní část algoritmu, kde se hledá laso mezi supergrafy. Nejprve se hledá mezi jednotlivými supergrafy a poté i mezi supergrafy, které jsou vzájemně kompozitní. Jestliže se pro alespoň jednu dvojici supergrafů nepodaří nalézt laso, je test jazykové inkluze nesplněn a algoritmus vrací FALSE, jinak je splněn a algoritmus vrací TRUE.

5 Návrh

Tato kapitola popisuje návrh nově vytvořeného rozšíření knihovny VATA pro Büchiho automaty. Jsou zde popsány datové struktury pro práci s těmito automaty a také popis implementace základních algoritmů jako jsou sjednocení, průnik, odstranění nedostupných a neukončujících stavů a reverze automatu.

5.1 Datové struktury pro explicitní kódování Büchiho automaty

V knihovně VATA je implementace dvou druhů konečně stavových automatů a to stromových a slovních. Tyto automaty se liší především ve zpracování přechodů a také tím, že stromové automaty nemají počáteční stavy. Büchiho automaty jsou velmi podobné konečným slovním automatům. Liší se pouze tím, že pracují s nekonečnými vstupními řetězci. Díky tomu mohou využívat velmi podobné ukládání přechodů jako u konečných automatů. K tomu slouží struktury, které budou popsány v následující části. Tyto struktury jsou velmi důležité z hlediska tvorby operací pro práci s Büchiho automaty a závisí na nich také efektivita těchto operací.

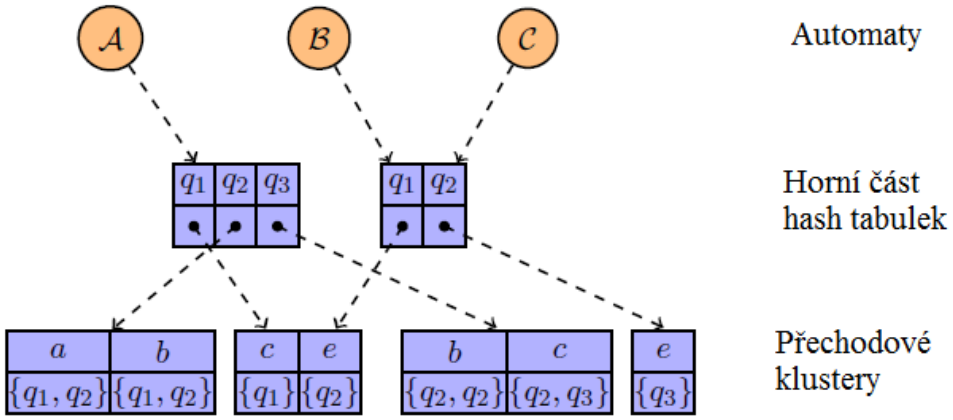
5.1.1 Analýza

Büchiho automaty jsou definovány pomocí množin stavů. Tyto stavy mohou být speciální, můžeme je rozdělit na počáteční a stavy s akceptační podmínkou (což jsou samozřejmě podmnožiny množiny všech stavů automatu). Je potřeba udržovat informace o počátečních stavech a také stavech s akceptační podmínkou, aby bylo možno rozlišovat je od stavů ostatních a také s nimi pracovat ve specifických operacích. Není však nutno ukládat všechny stavy samostatně, protože jsou dány implicitně stavy používanými uvnitř přechodů.

Přechody udržují většinu informací o BA a jsou často využívány při nejrůznějších operacích nad BA. Efektivita a výkon těchto operací silně závisí na efektivitě datových struktur pro uložení přechodů. Například, podobně je tomu u stromových automatů, není třeba uchovávat celou množinu stavů, ale efektivně ukládat přechody u těchto automatů.

5.1.2 Návrh datových struktur pro přechody u Büchiho automatů

Datové struktury pro ukládání přechodů automatu jsou založeny na hashovacích tabulkách. Jedná se o velmi podobný přístup jako u konečných automatů. Nejvrchnější hashovací tabulka mapuje daný stav na ukazatel na přechodový kluster. Přechodový kluster je také další hashovací tabulka, která mapuje symbol ze vstupní abecedy na množinu stavů, které jsou přístupné z daného stavu pomocí daného symbolu. Tuto datovou strukturu lze vidět na Obr. 5.1.



Obr. 5.1 - Datová struktura pro ukládání přechodů BA. Obrázek převzat z [10].

Datová struktura pro ukládání přechodů BA zjednodušuje tuto operaci u stromových automatů. Přechody u stromových automatů vypadají následovně: $q \xrightarrow{a} (q_1, \ldots, q_n)$, kde $q_1, q_2, \ldots, q_n$ jsou stavy stromového automatu a a je symbol vstupní abecedy. U BA vypadá přechod následovně: $q_1 \xrightarrow{a} q_2$, kde q_1, q_2 jsou stavy BA a symbol a je symbol vstupní abecedy BA. Lze tedy vidět již na první pohled rozdíl v těchto přechodech. Zatímco stromový automat musí ukládat celé n-tice, u BA tomu tak nutno není. Stromové automaty tedy kvůli efektivnosti uloží tyto n-tice pouze jednou z toho důvodu, že klustery mohou být velmi rozsáhlé a pro práci se zde využívá pouze ukazatel na kluster, namísto samotného klusteru. U BA však není výhodné pracovat s ukazatelem, protože zde nejsou žádné n-tice, ale pouze samotný stav. Je tedy nevhodné využít datové struktury jako množiny ukazatelů na n-tice, ale přímo množiny stavů automatu. Množiny stavů jsou přístupné přes přechodové klustery a obsahují všechny stavy přístupné z daného stavu v rámci určitého symbolu vstupní abecedy.

Další výhodou je, že množiny stavů nemusí být ve speciální množině odkazované přechodových klusterem, avšak mohou být přímo v tomto klusteru. Po aplikování této optimalizace mapuje přechodový kluster za pomocí symbolu přímo do množiny stavů dostupných pod tímto symbolem. Datové struktury pro stromové automaty a BA lze porovnat na Obr. 3.2 a Obr. 5.1.

Tyto datové struktury jsou efektivní z hlediska využití paměti, protože přechodové klustery jsou sdíleny mezi všemi BA. Nová tabulka se tvoří pouze tehdy, když je přidán nový prvek do jednoho z automatů.

5.2 Datové struktury pro počáteční stavy a akceptační podmínky

Z důvodu nutnosti při některých operacích je důležité rozlišovat dva druhy speciálních stavů. Jedná se o stavy počáteční a stavy s akceptační podmínkou automatu. Potřebujeme je při některých speciálních operacích rozlišit. Není však potřeba vytvářet nové speciální datové struktury, postačí již zmíněná hashovací tabulka. Počáteční stavy a stavy s akceptační podmínkou se ukládají do samostatných množin a ostatní stavy jsou součástí přechodů, proto žádnou samostatnou množinu nemají. Pro zjištění zda se stav nachází v množině počátečních stavů nebo stavů s akceptační podmínkou, se využívají různé metody.

5.3 Formát knihovny Timbuk

Knihovna VATA umožňuje načítat automaty z textové specifikace. Neexistuje však standardní formát pro BA, proto byla využita modifikace formátu knihovny Timbuk, stejně jako u konečných automatů. Formát knihovny Timbuk [11] se primárně využívá k popsání stromových automatů, ale může být také použit pro konečně stavové automaty, v našem případě pro Büchiho automaty. Je však třeba udělat několik modifikací. Tento formát se využívá jako vstupní formát všech automatů v knihovně VATA.

Příklad využití formátu knihovny Timbuk pro Büchiho automat lze vidět na Obr. 5.2. Na prvním řádku je popis vstupní abecedy i s její aritou. Symbol x má aritu nulovou a udává tak, že pokud se bude nacházet u přechodu, je stav na pravé straně přechodu stavem počátečním, $x \to s, s \in I$. To je nevýhoda u formátu knihovny Timbuk, poněvadž stromové automaty nemají počáteční stavy. Ostatní symboly budou mít vždy aritu 1. Arita nemusí být vůbec zadána a při konvertování automatu do vnitřní reprezentace se dosadí všem symbolům arita implicitně.

Na druhém řádku je název automatu. Na třetím se nachází seznam všech stavů automatu a na následujícím řádku se nachází seznam koncových stavů (v případě BA jsou to stavy s akceptační podmínkou).

Na dalších jsou přechody automatu. Na prvních z těchto řádků jsou vyznačeny počáteční stavy, a jak již bylo zmíněno výše, poznáme to podle symbolu x na levé straně přechodu. Lze uvést také příklad přechodu $s \xrightarrow{a} p$, ve formátu knihovny Timbuk je tento přechod zapsán následovně: $a(s) \to p$.

1. Ops $a:1\ x:0$
2. Automaton foo
3. States $s\ p\ q\ f$
4. Final States f
5. Transitions
6. $x \longrightarrow s$
7. $a(s) \longrightarrow p$
8. $a(s) \longrightarrow q$
9. $a(p) \longrightarrow f$
10. $a(q) \longrightarrow f$

Obr. 5.2 – Formát knihovny Timbuk pro Büchiho automat.

5.4 Překlad stavů a symbolů

Protože automat je zadán v textové podobě, je nutné jej převést do interní reprezentace, aby s ním bylo možno programově manipulovat. Konverze do této reprezentace je založena na mapování stavů ze vstupního formátu na celočíselný datový typ. Tento princip je také uplatněn pro symboly vstupní abecedy a lze jej vidět na Obr. 5.3. Tento mechanismus přináší efektivnější manipulaci se stavy a symboly při různých operacích s automaty. Další výhodou tohoto přístupu je sjednocení několika forem vstupů do jedné interní reprezentace.

Když jsou prováděny některé operace nad BA, je někdy potřeba serializovat výstup tohoto automatu. Symboly a stavy výsledného automatu jsou tak mapovány zpět do vstupního formátu využitím hashovacích tabulek, ve kterých je uloženo mapování. Díky tomuto přístupu je zajištěno, že výstup serializace je lépe čitelný, protože je zachována původní notace automatu.

q_1	q_2	q_3	q_4
1	2	3	4

a	b	c	d
1	2	3	4

Obr. 5.3 - Demonstrace překladu stavů a symbolů do interní reprezentace. Levá tabulka popisuje mapování stavů automatu na celá čísla a pravá tabulka mapuje symboly vstupní abecedy na celá čísla. Obrázek převzat z [10].

5.5 Algoritmy pro základní operace

V této části se nachází popis základních algoritmů pro implementaci základních operací, jako jsou sjednocení, průnik a odstranění neukončujících a nedostupných stavů.

5.5.1 Sjednocení

Sjednocení dvou BA $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \Delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}})$ a $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \Delta_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}})$ bylo popsáno v sekci 2.4.1 a je tvořeno podle následujícího algoritmu, který je shodný jako pro sjednocení konečných automatů. Využívá se zde tzv. reindexování, kdy se vytvoří index, který mapuje celočíselné hodnoty reprezentující stavy ze vstupních automatů na nové celočíselné hodnoty reprezentující ty samé stavy, ale v nově vzniklém automatu při této operaci. Avšak pokud se operace sjednocení provádí se vstupními automaty, jejichž množiny stavů jsou disjunktní, tak se reindexace nevyužívá.

Při operaci sjednocení se vytvoří prázdný automat, který je výsledným automatem sjednocení dvou předchozích automatů. Množiny počátečních stavů a stavů s akceptační podmínkou jsou zkopírovány do výsledného automatu z obou automatů. Dále všechny přechody automatů $\mathcal{A}$ a $\mathcal{B}$ jsou vloženy do výsledného automatu.

Při vkládání stavů do výsledného automatu se využívá reindexace. Důvodem je nutnost rozlišovat stavy dvou automatů. Pokud uvedeme příklad tak automaty $\mathcal{A}$ a $\mathcal{B}$ mohou mít stavy se stejnými celočíselnými hodnotami nebo stavy se stejnými jmény. Reindexace zajistí čitelnější výstup a rozliší tyto stavy ve výstupním automatu. Pokud uvedeme příklad: máme stav p automatu $\mathcal{A}$ a stav p automatu $\mathcal{B}$ ve výsledném automatu jsou tyto stavy pojmenovány jako p_1 a p_2.

5.5.2 Průnik

Operace průniku mezi dvěma BA $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \Delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}})$ a $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \Delta_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}})$ byla popsána v sekci 2.4.1. Definujeme dvojí stav $(p, q) \in \mathcal{A} \cap \mathcal{B}$, pro symbol $a \in \Sigma$ jako:
$\Delta(p, q) = \{(p', q') \mid \exists a \in \Sigma: (p, a, p') \in \Delta_{\mathcal{A}}, (q, a, q') \in \Delta_{\mathcal{B}}\}$.

Při konstrukci automatu $\mathcal{C}$ tak, aby akceptoval jazyk $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$ je nutno zaručit, že $\mathcal{C}$ navštíví nekonečněkrát alespoň některé akceptující stavy z $\mathcal{A}$ i $\mathcal{B}$. Výsledný automat $\mathcal{C}$ poté funguje tak, že nejprve setrvá, až při čtení libovolného slova navštíví nekonečněkrát akceptující stav z $\mathcal{A}$ a poté čeká na průchod akceptujícím stavem v $\mathcal{B}$ a zase se vrátí k čekání na průchod akceptujícího stavu z $\mathcal{A}$ atd. Akceptující stavy ve výsledném automatu tedy budou korespondovat se stavy z automatu $\mathcal{B}$.

5.5.3 Reverze

Reverzí BA $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}})$ se rozumí pouze prohození množiny počátečních stavů, stavů s akceptační podmínkou a otočením všech přechodů. Ty se otáčí tak, že přechod $p \xrightarrow{x} q \in \delta_{\mathcal{A}}$ je vložen do $\delta_{\mathcal{A}rev}$ ve formátu $q \xrightarrow{a} p$. Po těchto dílčích operací vznikne nový automat $\mathcal{A}_{rev} = (Q_{\mathcal{A}}, \Sigma, \delta_{\mathcal{A}rev}, F_{\mathcal{A}}, I_{\mathcal{A}})$.

Vstup : $BA\ \mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \Delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}})a\ BA\ \mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \Delta_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}})$
Výstup: $BA\ \mathcal{A} \cap \mathcal{B} = (Q_{\mathcal{A}\cap\mathcal{B}}, \Sigma, \Delta_{\mathcal{A}\cap\mathcal{B}}, I_{\mathcal{A}\cap\mathcal{B}}, F_{\mathcal{A}\cap\mathcal{B}})$
$Stack := \emptyset;$
$Reachable := \emptyset;$
$numb = 1;$
foreach $(p_{\mathcal{A}}, p_{\mathcal{B}}) \in I_{\mathcal{A}} \times I_{\mathcal{B}}$ **do** {
    $Vlož\ (p_{\mathcal{A}}, p_{\mathcal{B}})\ do\ I_{\mathcal{A}\cap\mathcal{B}};$
    if $\big((p_{\mathcal{A}}, p_{\mathcal{B}}) \notin Reachable\big)$**then** {
        $Vlož\ (p_{\mathcal{A}}, p_{\mathcal{B}})\ do\ Reachable;$
        $Vlož\ na\ Stack\ ((p_{\mathcal{A}}, p_{\mathcal{B}}), numb);$
    }
}
while $(Stack \neq \emptyset)$**do** {
    $Vem\ a\ odstraň\ dvojí\ stav\ (p_{\mathcal{A}}, p_{\mathcal{B}})\ z\ Stack.first;$
    $numb := Stack.second;$
    foreach $(q_{\mathcal{A}}, q_{\mathcal{B}}) \in \Delta(p_{\mathcal{A}}, p_{\mathcal{B}}), \forall a \in \Sigma$ **do** {
        if $(numb = 1)$**then** {
            if $(q_{\mathcal{A}} \in F_{\mathcal{A}})$ **then**
                $numb = 2;$
        }
        else {
            if $(q_B \in F_B)$ **then** {
                $numb = 1;$
                $Vlož\ (q_{\mathcal{A}}, q_{\mathcal{B}})\ do\ F_{\mathcal{A}\cap\mathcal{B}};$
            }
        }
        $Vlož\ (p_{\mathcal{A}}, p_{\mathcal{B}}) \xrightarrow{a} (q_{\mathcal{A}}, q_{\mathcal{B}})\ do\ \Delta_{\mathcal{A}\cap\mathcal{B}};$
        if $\big((q_{\mathcal{A}}, q_{\mathcal{B}}) \notin Reachable\big)$**then** {
            $Vlož\ (q_{\mathcal{A}}, q_{\mathcal{B}})\ do\ Reachable;$
            $Vlož\ na\ Stack\ ((q_{\mathcal{A}}, q_{\mathcal{B}}), numb);$
        }
    }
}
$Vrať\ \mathcal{A} \cap \mathcal{B} = (Reachable, \Sigma, \Delta_{\mathcal{A}\cap\mathcal{B}}, I_{\mathcal{A}\cap\mathcal{B}}, F_{\mathcal{A}\cap\mathcal{B}});$

Algoritmus 5.1: Algoritmus výpočet průniku dvou BA.

5.5.4 Odstranění nedostupných stavů

Máme-li BA $\mathcal{A}$, odstraněním nedostupných stavů získáme automat $\mathcal{B}$. Algoritmus odstranění těchto stavů je následující a zobrazuje jej Algoritmus 5.2. Postupně jsou prohledávány stavy od začátku, jestli jsou dosažitelné z počátečních stavů pro libovolné slovo $w \in \Sigma^*$. Pokud ano, uloží se do speciální množiny. Poté, všechny přechody dosažitelných stavů na levé straně přechodu, jsou vloženy do výsledného automatu. Jetliže je dosažitelný stav stavem s akceptační podmínkou, je stav vložený do výsledného automatu také stavem s touto vlastností. Počáteční stavy jsou vloženy do výsledného automatu.

5.5.5 Odstranění neukončujících stavů

Neukončující stavy jsou odstraněny z automatu $\mathcal{A}$ tak, že se nejprve odstraní nedostupné stavy. Poté je automat revertován a opět jsou z tohoto revertovaného automatu odstraněny nedostupné stavy. Revertovaný automat se opět revertuje a takto je dosaženo odstranění neukončujících stavů.

Vstup: $BA\ \mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \Delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}})$
Výstup: $BA\ \mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \Delta_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}})$
$I_{\mathcal{B}} := I_{\mathcal{A}}$;
$Reachable := I_{\mathcal{A}}$;
$Stack := Reachable$;
while $(Stack \neq \emptyset)$ **do** {
    $Vem\ a\ odstraň\ stav\ p\ ze\ Stack$;
    $pro\ všechny\ q \in \delta(p, a), \forall a \in \Sigma\ dělej$ {
        if $(q \in F_{\mathcal{A}})$ **then** $vlož\ q\ do\ F_{\mathcal{B}}$;
        $Vlož\ \{(q \xrightarrow{a} q') | q' \in Q_{\mathcal{A}}. \exists a \in \Sigma.\ q \xrightarrow{a} q' \in \Delta_{\mathcal{A}}\}\ do\ \Delta_{\mathcal{B}}$;
        if $(q \notin Reachable)$ **then** $vlož\ q\ do\ Stack\ a\ Reachable$;
    }
}
$vrať\ \mathcal{B} = (Reachable, \Sigma, \Delta_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}})$;

Algoritmus 5.2: Algoritmus pro odstranění nedostupných stavů BA.

6 Implementace

Tato kapitola popisuje nově přidaný modul do knihovny VATA pro práci s Büchiho automaty. Nově přidaný modul je stejně jako knihovna VATA psán v jazyce C++. Nejprve jsou popsány již existující moduly využité knihovny VATA, poté načítání a manipulace přímo s Büchiho automatem. Na závěr této kapitoly je popsána implementace operace testování jazykové inkluze pro Büchiho automaty.

6.1 Použité moduly z knihovny VATA

Pro práci s Büchiho automaty v knihovně VATA můžeme využít několik již hotových modulů. Tyto moduly poslouží k vytvoření modulu pro práci s Büchiho automaty. V této části se nachází seznam těchto použitých modulů, a jak byly využity.

6.1.1 Parser a serializér

Pro načítání automatů z textové specifikace jsou využity dva moduly. Jeden pro parsování vstupu, který se nazývá *parser*, a druhý pro serializaci nazývaný *serializér*. Je to z toho důvodu, že všechny automaty jsou načítány ze stejného vstupního formátu a je tedy vhodné tyto dva implementované moduly využít. *Parser* se využívá k převodu vstupního formátu do interní reprezentace a v případě Büchiho automatů do explicitního kódování. Jestliže chceme interní reprezentaci převést na výstupní formát, je využit *serializér*. Tento formát je shodný s formátem, se kterým pracuje *parser*.

6.1.2 Simulace

Další využití má modul pro výpočet relace simulace pro část se stromovými automaty. U Büchiho automatů je tento implementovaný modul využit pro výpočet simulace. Protože stromové automaty používají výpočet na LTS[1] tak, že převedou automat na LTS, musí se vytvořit nová konverze Büchiho automatů do LTS. Je zapotřebí tuto implementaci umístit do části knihovny pro práci s Büchiho automaty.

6.1.3 Utility

Knihovna VATA poskytuje velmi mnoho utilit, které jsou použitelné pro implementaci rozšíření pro Büchiho automaty. Tyto utility usnadňují práci díky tomu, že poskytují třídy pro snadnější zpracování Büchiho automatů. Díky tomu byl usnadněn vývoj nového modulu pro Büchiho automaty a také zajištěna soudržnost celé knihovny. Pokud uvedeme nějaký příklad, třída *AutDescription* pomáhá při reprezentaci automatu pro parsování. Dalšími příklady jsou třídy *TwoWayDict* a *TranslatorStrict*, které jsou použity při konverzi Büchiho automatu do explicitní reprezentace.

[1] LTS (labelled transition system) – obsahuje určitý počet stavů, které jsou spojeny hranami označených činností nějakého systému.

6.2 Načtení a manipulace s Büchiho automaty v explicitním kódování

Stěžejní částí tohoto rozšíření pro knihovnu VATA je načítání a manipulace s Büchiho automaty. K tomu slouží hlavní třída této implementace *ExplicitBuchiAut*, která reprezentuje pomocí Büchiho automatů explicitní reprezentaci. Tato třída obsahuje datové struktury pro explicitní reprezentaci, které byly popsány v předešlé kapitole. Třída obsahuje metody pro načítání automatu z textové specifikace a práci s parserem a serializérem. Dále je zde několik metod pro manipulaci s Büchiho automaty, jako je například testování, zda pro daný stav platí akceptační podmínka nebo nastavení, toho že je stav počáteční atd. Ještě může být zmíněno, že se zde nachází překlad stavů a symbolů na celočíselné hodnoty.

6.3 Překlad Büchiho automatu do LTS

Pokud chceme provést výpočet relace (maximální) simulace nad Büchiho automatem pomocí metody pro výpočet simulace na LTS, jež se nachází v knihovně VATA, je nezbytné, převést tento automat do LTS. To samozřejmě zahrnuje rozklad množiny stavů BA na dvě nebo tři třídy (akceptační, neakceptační a třídu reprezentující počáteční stavy) a inicializaci relace simulace. Tato operace je provedena za pomocí algoritmu, kde všechny přechody vstupního BA převedeme na hrany LTS. Současně je každý ze zpracovávaných stavů rozdělen do jednotlivých částí podle toho, zda se jedná o stav s akceptační podmínkou či nikoliv. Pokud se v automatu, který je simulován, nachází pouze stavy s akceptační podmínkou, vzniká jen jeden oddíl. Pokud jsou zde stavy bez akceptační podmínky, je vytvořen další oddíl s těmito stavy. Poté, co jsou zpracovány všechny přechody, je vytvořen oddíl pro počáteční stavy. Na závěr je relace simulace inicializována podle pravidel tak, že každý stav s akceptační podmínkou simuluje ostatní stavy s akceptační podmínkou. Dále se simulují mezi sebou ty stavy, které tuto podmínku nemají. Stavy s akceptační podmínkou simulují také stavy bez ní, avšak ne naopak. Vytvořené oddíly, LTS reprezentace a inicializace simulace jsou předány algoritmu pro výpočet simulace nad LTS.

6.4 Jazyková inkluze

Implementace testování jazykové inkluze dvou Büchiho automatů patří k nejnáročnějším úlohám této práce. K této operaci je využit přístup založený na Ramseyho větě (Ramsey-based), který je popsán v sekci 4.1. V této sekci se seznámíme s implementací tohoto algoritmu. Nejprve se strukturami pro uložení supergrafů, které jsou důležitými aspekty toho algoritmu, a dále s konkrétními částmi celého algoritmu, které byly využity k jeho implementaci. V následujícím popisu budeme pracovat se dvěma Büchiho automaty $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \Delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}})$ a $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \Delta_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}})$.

6.4.1 Návrh datových struktur

Důležitým aspektem při implementaci testování jazykové inkluze dvou Büchiho automatů jsou datové struktury. Hlavní datovou strukturou, se kterou se převážně pracuje, jsou supergrafy. Ty se skládají z grafu a hrany. Pokud bychom rozkládaly i tyto části na menší, hrany se skládají z libovolných dvou stavů p, q a symbolu a, kde $a \in \Sigma$ a $p, q \in Q_{\mathcal{A}}$. Graf se skládá z více hran, avšak liší se tím, že jeho stavy jsou $p_i, q_i \in Q_{\mathcal{B}}$ a symbol a patří do množiny $\{-1, 0, 1\}$. K reprezentaci grafu byla zvolena datová struktura std::vektor. Prvky grafu jsou dvojice stavů a symbol, který spojuje tyto stavy. Jedná se o vektor hran. Graf je zobrazen na Obr. 6.1. Na Obr. 6.2 je zobrazen příklad supergrafu.

Indexy	0	1	2	3	4	5
Stavy	(p,q)	(p,r)	(q,q)	(q,t)	(t,r)	(r,p)
Symbol přechodu	0	1	0	0	-1	1

Obr. 6.1 - Reprezentace grafu při testování jazykové inkluze. Skládá se z dvojice stavů a symbolu u přechodu mezi těmito dvěma stavy.

	0	1	2	3	4	5
Graf	(p,q)	(p,r)	(q,q)	(q,t)	(t,r)	(r,p)
	0	1	0	0	-1	1
Hrana	(p,q)					

Obr. 6.2 - Reprezentace supergrafu při testování jazykové inkluze. Skládá se z dvojice hrana a graf.

6.4.2 Algoritmus testování jazykové inkluze

V této části se zaměříme na jednotlivé aspekty algoritmu, založeném na Ramseyho větě k testování jazykové inkluze, který byl popsán v sekci 4.1.

Vstup a výstup

Vstupními prvky jsou dva BA $\mathcal{A}, \mathcal{B}$ a také množina supergrafů $S^1_{\mathcal{A},\mathcal{B}}$, která se musí nejprve vygenerovat ze vstupních automatů. Generování této množiny se provádí tak, že se nejprve vygenerují jednotlivé hrany podle automatu $\mathcal{A}$. Vytvořená množina hran se uloží tak, že jsou všechny hrany, které přecházejí mezi stavy se stejnými symboly, uloženy do std::vektoru na stejný index. Každý index tohoto vektoru obsahuje pole hran. Poté, jsou vygenerovány grafy

podle automatu $\mathcal{B}$. Každý vygenerovaný graf je tvořen podle jednoho symbolu vstupní abecedy. Symboly u jeho hran jsou z množiny $\{-1, 0, 1\}$. Posléze, co jsou vygenerovány hrany a jednotlivé grafy, jsou spojeny v supergrafy. Ke každému grafu jsou přidány hrany a to pouze ty, jejichž tvorba vznikla na základě stejných symbolů vstupní abecedy. Výstupem celého algoritmu je pak výroková hodnota $TRUE$, pokud byly splněny podmínky jazykové inkluze mezi $\mathcal{A}, \mathcal{B}$, což znamená, že platí $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$, jinak je vrácena hodnota $FALSE$.

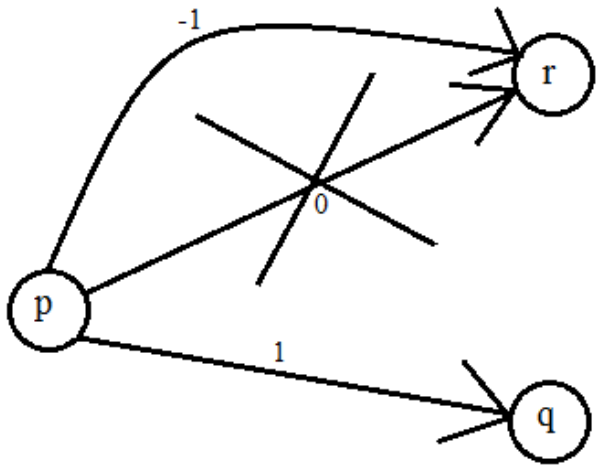
Příprava

V této části dochází k přípravě před hlavní částí algoritmu. Nejprve je vytvořena množina minimalizovaných supergrafů, což je $Next = \{Min_S(\boldsymbol{g}) | \, \boldsymbol{g} \in S^1_{\mathcal{A},\mathcal{B}}\}$. Je k tomu využita funkce Min_S, která je popsána níže. Z této množiny jsou vybírány supergrafy a testovány mezi sebou na to, zda jeden nepokrývá druhý tj. $\boldsymbol{f} \sqsubseteq_S^{\forall\exists} \boldsymbol{g}$. Tato operace se nazývá relace pokrytí a je popsána níže. Všechny supergrafy, u kterých bylo zjištěno, že mohly vzniknout z jiných supergrafů, jsou odstraněny a už se dále v algoritmu neuvažují.

Funkce Min a relace pokrytí

Minimalizace supergrafů je důležitá z hlediska efektivity Ramsey-based přístupu pro testování jazykové inkluze. Je k tomu využita funkce Min_S. Minimalizace supergrafu se provádí tak, že se prochází jeho stavy a hledá se, jestli z těchto stavů nevede více přechodů. Pokud ano, minimalizace vypadá následovně. Máme-li graf $\boldsymbol{g} = (\bar{g}, g)$ a libovolné stavy $p, q, r \in g$, pokud existují přechody $p \xrightarrow{a} r$ a $p \xrightarrow{a'} q$, kde $a \leq a'$, $r \preccurlyeq q$, poté můžeme přechod $p \xrightarrow{a} r$ odstranit a vytvořit nový $p \xrightarrow{a''} r$, kde $a'' \leq a$. Tato situace je zachycena na obrázku Obr. 6.3. Po odstranění všech takových přechodů u všech stavů grafu, vzniká nový menší supergraf $Min_S(\boldsymbol{g})$.

Další operace z hlediska efektivity je testování zda supergraf pokrývá jiný supergraf. Máme-li dva supergrafy $\boldsymbol{f}, \boldsymbol{g} \in S^1_{\mathcal{A},\mathcal{B}}$, jestliže je relace pokrytí mezi těmito supergrafy splněna, pak supergraf $\boldsymbol{g}$ pokrývá supergraf $\boldsymbol{f}$ a nemusíme jej nadále vůbec uvažovat, protože chceme pracovat pouze s těmi nejmenšími. Menší grafy mají menší jazyk a tudíž větší šanci nalézt protipříklad k universalitě.



Obr. 6.3 - Ukázka odstranění přechodu u grafu při minimalizaci supergrafu.

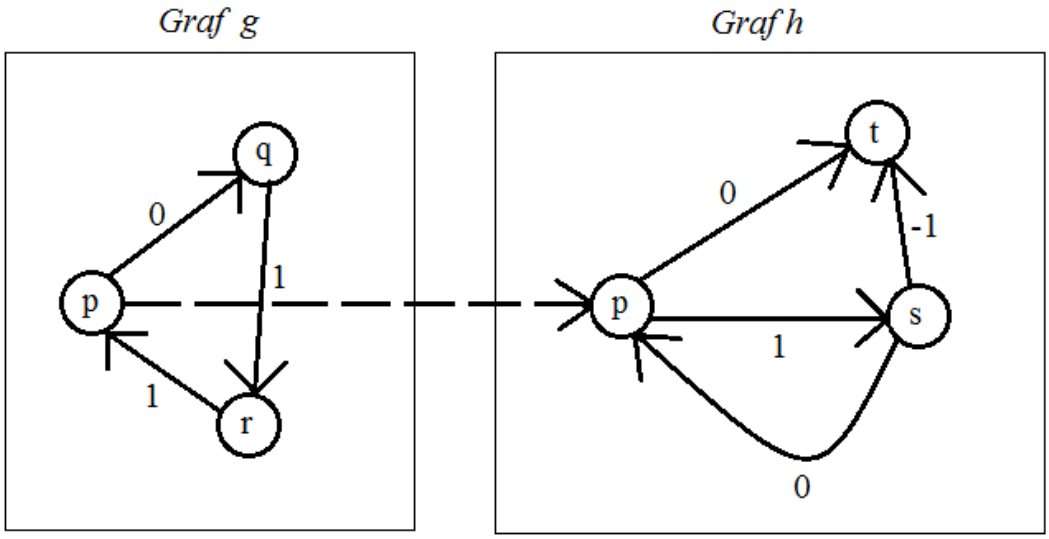
Hlavní část algoritmu

Po přípravě se přechází k hlavní části algoritmu testování jazykové inkluze. Ta se skládá z několika dílčích částí a to převážně testování dalších vlastností supergrafů. Důležitými částmi jsou testování $RDGT$ (relaxed double-graph test) a testování kompozice. $RDGT$ je složitější operace, proto je popsána níže. Algoritmus využívá kompozici dvou supergrafů, proto se musí nejprve otestovat, zda je možné vytvořit jejich kompozici. Testování kompozice je splněno, jestliže existuje stejný stav ve dvou supergrafech. Požadovaný stav je dosažitelný z počátečního stavu automatu $\mathcal{A}$ v prvním supergrafu a v druhém, se z tohoto stejného stavu dá pokračovat, tj. máme dva supergrafy $\boldsymbol{f}, \boldsymbol{g} \in S^1_{\mathcal{A},\mathcal{B}}$ a přechod $p \xrightarrow{a} q \in g$, pak existuje $q \xrightarrow{a} r \in h$. Že jsou dva supergrafy kompozitní se značí $(\boldsymbol{g}; \boldsymbol{h})$. Pokud vrátí operace $RDGT$ hodnotu $FALSE$, tak neplatí jazyková inkluze mezi dvěma vstupními automaty. Naopak pokud hodnota $FALSE$ u operace nenastane, je vrácena hodnota $TRUE$ a je tak dokázáno, že platí $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$.

RDGT

Relaxed double-graph test je stěžejní částí algoritmu přístupu Ramsey-based pro testování jazykové inkluze. Vstupem tohoto testu jsou dva supergrafy, tj. $RDGT(\boldsymbol{g}, \boldsymbol{h})$ a výstupem výroková hodnota $TRUE$ nebo $FALSE$. Je to rozhodnuto na základě dvou podmínek, kdy stačí, aby platila pouze jedna z nich. První z nich je, že množina $Z_{\boldsymbol{g},\boldsymbol{h}}$ není slabě správně kompozitní. Tato vlastnost platí pro hrany mezi supergrafy. Máme-li dvě hrany $\bar{g} = < p, q >$ a $\bar{h} = < q_1, q_2 >$, kde $p \in I_{\mathcal{A}}$ a $q_2 \in F_{\mathcal{A}}$, pak množina $Z_{\boldsymbol{g},\boldsymbol{h}}$ je slabě správně kompozitní. To nesmí nastat, pokud by nebyla splněna další část $RDGT(\boldsymbol{g}, \boldsymbol{h})$ a laso hledající test $LFT(g, h)$. V tomto testu se pracuje s grafy obou supergrafů a hledá se tzv. laso. Jedná se o to, že z prvního grafu g se lze dostat do stavu, např. p, který je obsažen v obou grafech, $p \in g$ i h. Jakmile je takový stav nalezen, hledá se laso v grafu h. Hledání lasa je popsáno níže. Na Obr. 6.4 je tato situace stručně znázorněna. Pokud je laso nalezeno nebo množina $Z_{\boldsymbol{g},\boldsymbol{h}}$ není slabě správně kompozitní, vrací $RDGT(\boldsymbol{g}, \boldsymbol{h})$ hodnotu $TRUE$, v opačném případě se

vrací $FALSE$. Platí $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$, právě tehdy, když je splněno $RDGT(\boldsymbol{g}, \boldsymbol{h})$ pro všechny $\boldsymbol{g}, \boldsymbol{h} \in S^{f}_{\mathcal{A},\mathcal{B}}$.



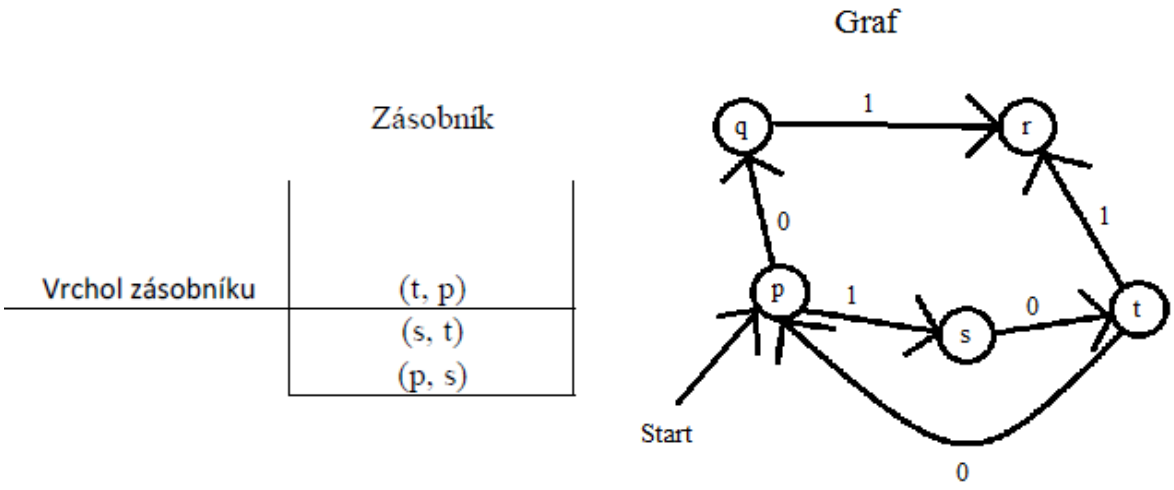
Obr. 6.4 - Ukázka LFT(g,h). Graf g udává „rukojeť“ a graf h je „laso“.

Laso hledající test

Laso hledající test je součástí testu $RDGT(\boldsymbol{g}, \boldsymbol{h})$. Jedná se o hledání silně souvislé komponenty[2]. K tomuto hledání je využito prohledávání do hloubky se zpětným návratem, tzv. bactracking. Pro nalezení všech silně souvislých komponent existuje algoritmus tzv. Tarjanův algoritmus[3]. V tomto případě byl použit pouze z části. Tarjanův algoritmus pro procházení stavy grafů, využívá rekurzi a také číslování jednotlivých stavů, avšak v této práci tomu tak není. Rekurze je nahrazena zásobníkem. Stavy jsou na zásobník ukládány po dvojicích. Dvojice stavů uložená na vrcholu je testována, zda pro stav na pravé straně z této dvojice, existuje v tomto zásobníku dvojice, kde je tento stav na levé straně. Z toho vyplývají dvě možné varianty pokračování a to pokud je stav nalezen nebo ne. Pokud nastane druhá varianta, je vygenerována další dvojice stavů, kterou se pokračuje v zanořování. Jakmile se nedá pokračovat v zanořování, je tato dvojice vyjmuta. Jestliže je stav nalezen, pak se v zásobníku nachází silně souvislá komponenta, tedy laso a $LFT(g, h)$ je splněno. Na Obr. 6.5 je příklad grafu a zásobníku, ve chvíli kdy je v něm nalezeno laso.

[2] Silně souvislá komponenta je podgraf, v němž pro všechny dvojice vrcholů $p, q \in h$ existuje cesta z p do q a zároveň z q do p.

[3] Tarjanův algoritmus je algoritmus hledající silně souvislé komponenty v grafu.



Obr. 6.5 - Obrázek se zachycením situace nalezení lasa. Vlevo se nachází zásobník se stavy, které obsahuje na konci při prohledávání grafu vpravo.

7 Experimentální výsledky

Tato kapitola popisuje výsledky experimentů, prováděných nad implementací algoritmu pro testování jazykové inkluze. Jedná se o experimenty nad implementací přístupu, založeného na Ramseyho větě. Experimenty byly prováděny na osobním laptopu se systémem Debian 64bit Linux, 2x Intel Core i5 (2,6 GHz, 2 jádra a 3 MB cache) a 4GB RAM.

7.1 Výsledky algoritmu založené na Ramseyho větě

Implementace testování jazykové inkluze byla testována a výsledky testů byly porovnány s nástrojem RABIT. Ten byl popsán v sekci 3.1.3.

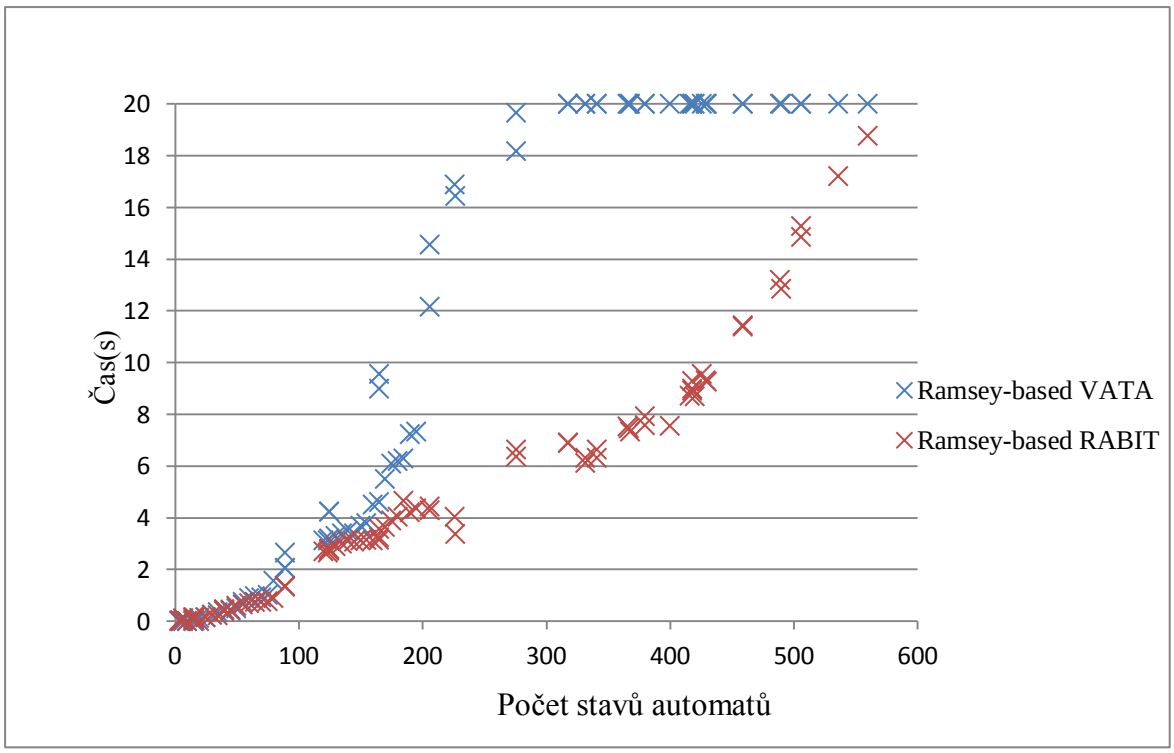
7.1.1 Porovnání s nástrojem RABIT

V této sekci se nachází vyhodnocení a porovnání výsledků testování jazykové inkluze implementací pro Büchiho automaty a nástroje RABIT. Nástroj RABIT obsahuje několik optimalizací, které zvyšují efektivitu testování jazykové inkluze. K té využívá optimalizovaný přístup založený na Ramseyho větě. Výsledky experimentů implementace nacházející se v prezentované práci, byly porovnány s nástrojem RABIT, a to ve dvou variantách.

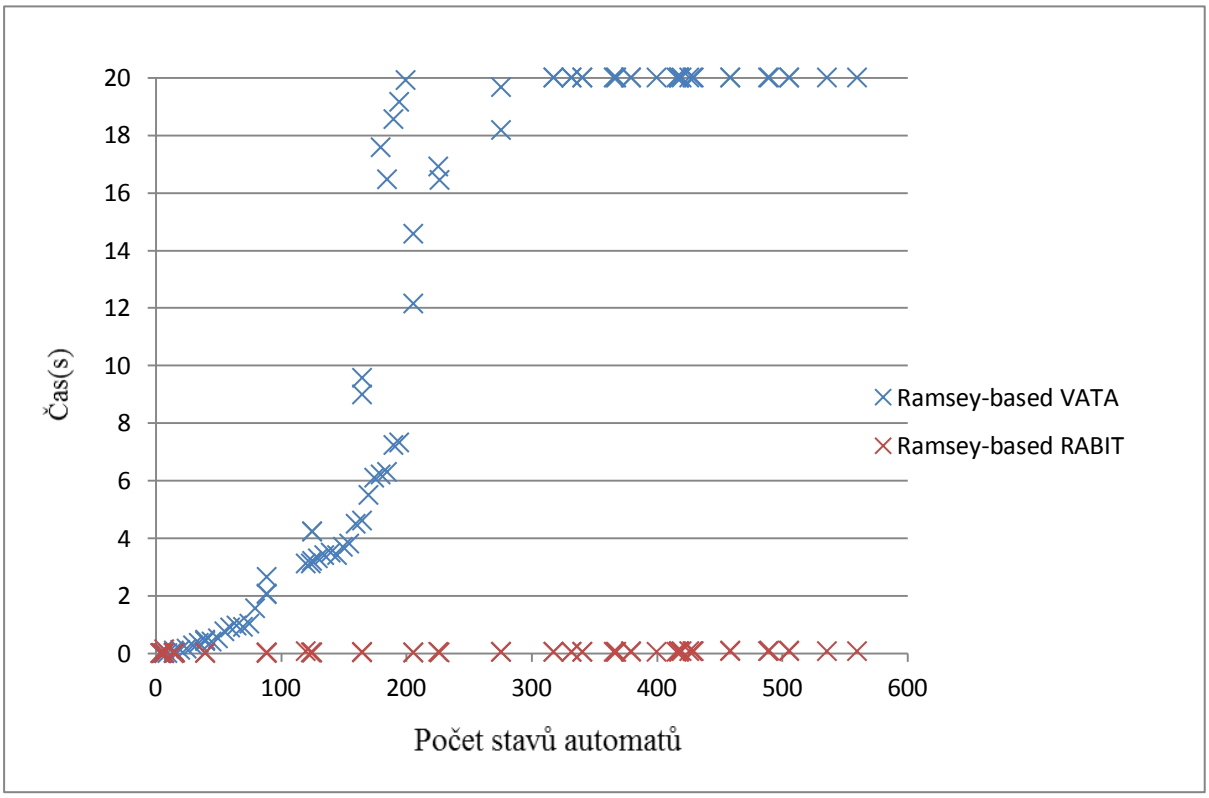
Nejprve se jedná o porovnání s nástrojem RABIT, kdy nebyly při experimentech využity žádné optimalizace. Po porovnání výsledků testů nad vstupními automaty s nižšími počty stavů, se výsledky obou implementací téměř shodují. Jakmile však začne počet stavů testovaných automatů vzrůstat, roste také časová náročnost operace testování jazykové inkluze prezentované v této práci. Je to způsobenou neefektivitou datových struktur pro ukládání supergrafů. Z grafu nacházejícím se na Obr. 7.1 lze vidět, jak vzrůstá časová náročnost při růstu počtu stavů vstupních automatů.

Při dalším porovnání byl nástroj RABIT spuštěn s několika optimalizacemi. Jedná se například o využití minimalizace vstupních automatů nebo při simulaci pokrytí použití dopředného vyhledání atd. Tato varianta experimentu poukazuje na to, že pokud spustíme nástroj RABIT se všemi jeho optimalizacemi, lze dosáhnout velmi rychlého provádění algoritmu testování jazykové inkluze. Při porovnání s implementací ve zde prezentované práci, je patrný velký rozdíl v časové náročnosti operace, což je ukázáno na Obr. 7.2.

Testování jazykové inkluze mezi dvěma BA je výpočetně velmi náročný problém, proto je potřeba využít optimalizované přístupy. Ve zde prezentované práci byl využit přístup založený na Ramseyho větě. Výsledky experimentů prokázaly, že ani optimalizovaný algoritmus nemusí stačit k zajištění efektivity prováděné operace. Je to způsobeno neoptimalizovanou manipulací se supergrafy.



Obr 7.1: Graf zobrazující porovnání výsledky testování implementace jazykové inkluze pro Büchiho automaty a nezrychlenou verzi testování jazykové inkluze nástroje RABIT.



Obr 7.2: Graf zobrazující porovnání výsledky testování implementace jazykové inkluze pro Büchiho automaty a zrychlenou verzi testování jazykové inkluze v nástroji RABIT.

8 Závěr

V této práce bylo vytvořeno rozšíření knihovny VATA o podporu práce s nedeterministickými Büchiho automaty, které bývají často využívány ve formální verifikaci. Rozšíření podporuje několik základních operací jako je sjednocení a průnik dvou BA, odstranění nedostupných a neukončujících stavů automatu atd. Poskytuje však také implementaci složitější operace, a to algoritmus pro testování jazykové inkluze mezi dvěma BA.

Pro tvorbu datových struktur Büchiho automatů byla využita explicitní reprezentace, která byla implementována modifikací datových struktur pro konečné automaty. Datové struktury umožňují jednoduše modifikovat automaty a provádět tak nad nimi základní operace. VATA mimo jiné umožňuje snadnou integraci dalších rozšíření, protože je psána modulárně.

Testování jazykové inkluze je velmi výpočetně náročný problém, proto bylo vyvinuto několik optimalizovaných algoritmů, řešící tento problém. V této práci byl využit optimalizovaný přístup založený na Ramseyho větě, pracující se supergrafy.

Výsledky experimentů nad implementací testování jazykové inkluze prokázaly, že se jedná o velmi časově náročnou operaci. Docházelo k velkým časovým prodlevám při vykonávání algoritmu testování jazykové inkluze, které byly způsobeny neefektivitou datových struktur pro ukládání a manipulaci se supergrafy.

Směr, kterým by se nadále práce mohla vyvíjet, by mohlo být vytvoření nových datových struktur, pro efektivnější manipulaci se supergrafy. Ty jsou využity při testování jazykové inkluze založeném na Ramseyho větě. Dalšími zajímavými směry by mohly být např. přidání algoritmu Rank-based pro testování jazykové inkluze mezi dvěma Büchiho automaty nebo rozšíření knihovny VATA o podporu práce s ostatními známými druhy ω-automatů jako např. Mullerovi, Streettovy či Rabinovy automaty.

Bibliografie

[1] Julius Richard Büchi. On a decision method in restricted second order arithmetic. In *Proc. International Congress on Logic, Method and Philosophy of Science*. Stanford University Press, 1960. S. 1-12.

[2] Berndt Farwer. ω-Automata. In *A Guide to Current Research*, Springer Berlin, 2002. S. 3-21. ISBN 978-3-540-36387-3.

[3] Ondřej Lengál, Jiří Šimáček a Tomáš Vojnar. VATA: A library for Efficient Manipulation of Non-deterministic Tree Automata. In *Proc. of TACAS 2012*, volume 7214. Springer-Verlag, 2012. S. 79-94.

[4] A. Duret-Lutz a D. Poitrenaud. Modeling, Analysis, and Simulation of Computer and Telecommunications Systems. In *Proc. The IEEE Computer Society's 12th Annual International Symposium on.* MASCOTS, 2004. S. 76-83. ISSN 1526-7539.

[5] Yih-Kuen Tsay, Yu-Fang Chen, Ming-Hsien Tsai, Kang-Nien Wu a Wen-Chin Chan. GOAL: A Graphical Tool for Manipulating Büchi Automata and Temporal Formulae. In *Proc. of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 2007. S. 466-471.

[6] Parosh Aziz Abdulla, Yu-Fang Chen, Chih-Duo Hong, Richard Mayr, Lorenzo Clemente, Tomáš Vojnar, Lukáš Holík, Ondřej Lengál a Jiří Šimáček. Nástroj RABIT [online]. Dostupné na: <http://www.languageinclusion.org/doku.php?id=tools>. Poslední změna 27. 2. 2013 [cit. 20. 1. 2014].

[7] Orna Kupferman a Moshe Y. Vardi. Weak Alternating Automata Are Not that Weak. *ACM Transactions on Computanional Logic*. Volume 2. 2001. S. 408-429.

[8] Parosh Aziz Abdulla, Yu-Fand Chen, Lorenzo Clemente, Lukáš Holík, Chih-Duo Hong, Richard Mayr a Tomáš Vojnar. Simulation Subsumption in Ramsey-Based Büchi Automata Universality and Inclusion Testing. In. *Proc. 22nd International Conference, CONCUR 2011*, volume 6174. Springer Berlín, 2010. S. 132-147.

[9] Parosh Aziz Abdulla, Yu-Fand Chen, Lorenzo Clemente, Lukáš Holík, Chih-Duo Hong, Richard Mayr and Tomáš Vojnar. Advanced Ramsey-Based Büchi Automata Inclusion Testing. In *Proc. 22nd International Conference, CONCUR 2011*, volume 6901. Springer Berlin, 2011. S. 187-202.

[10] Milan Hruška. *Efficient Algorithms for Finite Automata*. Brno: VUT FIT v Brně, 2013. Bakalářská práce.

[11] Webová stránka knihovny Timbuk. Timbuk[online]. Dostupná na: <http://www.irisa.fr/celtique/genet/timbuk/>, Poslední změna 2012 [cit. 29. 1. 2014].

[12] Webová stránka knihovny VATA. Knihovna VATA[online]. Dostupná na: <http://www.fit.vutbr.cz/research/groups/verifit/tools/libvata/>, Poslední změna 2013 [cit. 2014-1-20].

[13] Webová stránka nástroje GOAL. GOAL[online]. Dostupný na: <http://goal.im.ntu.edu.tw/wiki/doku.php>, Poslední změna 11. 5. 2014 [cit. 30. 1. 2014].

Seznam příloh

Na přiloženém CD se nachází:

- implementace knihovny VATA s rozšířením o práci s Büchiho automaty,
- soubor čti mě, ve kterém se nachází návod k instalaci a spuštění implementace,
- elektronická podoba této práce.